UNIVERSIDADE FEDERAL DE MINAS GERAIS
DEPARTAMENTO DE ENGENHARIA ELETRÔNICA

Apostila sobre Introdução ao Projeto com Microcontroladores e Programação de Periféricos

Disciplina de Laboratório de Sistemas, Processadores e Periféricos

baseado no microcontrolador PIC32MX360F512L
e no kit de desenvolvimento EXPLORER16BR da *Labtools*.

Autores:

Marconi de Oliveira Júnior
marconioliveirajr@gmail.com
Engenheiro Eletricista
Universidade Federal de Minas Gerais

Professor Ricardo de Oliveira Duarte
ricardoduarte@ufmg.br
Departamento de Engenharia Eletrônica
Universidade Federal de Minas Gerais

Versão: Dezembro de 2010

Sumário

1. Introdução 6
2. Microcontroladores, microprocessadores e sistemas embarcados 9
3. Material utilizado no curso 10
4. MPLAB® IDE e MPLAB® C32 12
4.1. Instalação MPLAB® IDE 12
4.2. Instalação MPLAB® C32 19
5. Criação de Projeto 22
6. Gravação do Projeto no PIC 27
7. Atividades Práticas 30
7.1. Aula 1 – Configuração de Pinos de Entrada e Saída 31
7.2. Aula 2 – Configuração do Periférico de Interrupções de Eventos Externos por Notificação de Mudança de Estado. 37
7.3. Aula 3 - Configuração de Periféricos Contadores (Temporizadores ou *Timers* e Contadores de eventos externos ou *Counters*). 44
7.4. Aula 4 – Configuração de Interface para Comunicação com Displays de Cristal Líquido (LCDs) alfa numéricos. 53
7.5. Aula 5 – Configuração de Conversores Analógicos/Digitais (Conversores A/D) 62
7.6. Aula 6 – Configuração de Interface para Comunicação serial UART com RS-232 71
7.7. Aula 7 – Configuração do Periférico de Modulação de Largura de Pulsos (PWM) 81
ANEXOS 86
Anexo I : Adicionando arquivos durante a criação do projeto 87
Anexo II : Adicionando arquivos depois da criação do projeto 91
Anexo III : Utilização do Socket para Gravação/Depuração 92
Anexo IV : MPLAB® SIM 93
Watchpoint 97
Estímulo 98
Logic Analyzer 102
Stopwatch 106
ERRATAS 110
Errata I : Esquemático Leds e Botoes 111
Errata II : Efeito de carga do ICD2BR 114
Referências Bibliográficas 115

Ilustrações

Figura 1 Explorer 16 BR 10
Figura 2 ICD2 BR 10
Figura 3 Plugin PIC32MX360F512L 10
Figura 4 Conector RJ12 11
Figura 5 Conector USB 11
Figura 6 Fonte Alimentação 11
Figura 7 CD Explorer 16BR 12
Figura 8 Pastas CD Explorer16BR 12
Figura 9 MPLAB® IDE - instalação 13
Figura 10 MPLAB® IDE – termo de compromisso 13
Figura 11 MPLAB® IDE - modo de instalação 14
Figura 12 MPLAB® IDE - local de instalação 14
Figura 13 MPLAB® IDE – termo de compromisso 2 15
Figura 14 MPLAB® IDE – termo de compromisso 3 15
Figura 15 MPLAB® IDE – revisão da instalação 16
Figura 16 MPLAB® IDE – Status da instalação 16
Figura 17 MPLAB® IDE – compilador HI-TECH C 17
Figura 18 MPLAB® IDE - fim da instalação 17
Figura 19 MPLAB® IDE - executável 18
Figura 20 MPLAB® C32 - boas vindas 19
Figura 21 MPLAB® C32 - termo de compromisso 19
Figura 22 MPLAB® C32 - local de instalação 20
Figura 23 MPLAB® C32 – instalação 20
Figura 24 MPLAB® C32 - progresso da instalação 21
Figura 25 MPLAB® C32 - fim da instalação 21
Figura 26 Localização MPLAB® 22
Figura 27 MPLAB® IDE 22
Figura 28 Criação de Projetos - Tela de Boas Vindas 23
Figura 29 Criação de Projetos - Seleção do Dispositivo 23
Figura 30 Criação de Projetos - Seleção do Compilador 24
Figura 31 Criação de Projetos - Nome e Localização do Projeto 24
Figura 32 Criação de Projetos - Adição de Arquivos 25
Figura 33 Criação de Projetos - Resumo 26
Figura 34 Criação de Projetos - Workspace 26
Figura 35 Seleção do PIC 28
Figura 36 Conexão ICD 28
Figura 37 Warning ICD2 29
Figura 38 ICD2 Conectado 29
Figura 39 Compilação correta do programa 35
Figura 40 ICD2BR Corretamente conectado 35
Figura 41 Gravação concluída com sucesso 36
Figura 42 Circuito de Clock do Timer 48
Figura 43 Overload do registrador PR1 52
Figura 44 LCD alfa-numérico 16x2 53
Figura 45 Inicialização do display HDM16216H-B (obtida no datasheet do LSD) 54
Figura 46 Comandos do LCD 55
Figura 47 Endereço das células do LCD 16x2 55
Figura 48 Conversão A/D - D/A 63
Figura 49 Circuito de amostragem do sinal 64
Figura 50 Sequencia de amostragem e conversão do ADC 64

Figura 51 Esquema elétrico do trimpot do kit EXPLORER16BR para utilizacão do conversor AD 68
Figura 52 Diagrama em blocos módulo AD 68
Figura 53 Formato do dado convertido 69
Figura 54 Esquema elétrico do sensor de temperatura do kit EXPLORER16BR para utilizacão do conversor AD 70
Figura 55 Conexão cabo DB9 71
Figura 56 Conector DB9Macho/DB9Femea (esquerda), DB9MAcho/USB (direita). 71
Figura 57 Esquema elétrico do conector DB9 e Driver MAX232 do kit EXPLORER16BR para comunicação RS232. 72
Figura 58 Exemplo de cálculo do Baud Rate 76
Figura 59 Hyperterminal 77
Figura 60 Hyperterminal 77
Figura 61 Parâmetros da comunicação RS232. 78
Figura 62 Finalizar comunicação RS232. 78
Figura 63 Comunicação RS232. Aula 6 - Atividade 1. 78
Figura 64 Tela inicial programa de senha. 79
Figura 65 Tela LCD programa de senha. 79
Figura 66 Senha correta. 80
Figura 67 Senha incorreta 80
Figura 68 Geração do sinal PWM 81
Figura 69 Diagrama em blocos do módulo output compare 82
Figura 70 Anexo I - Criação de Pasta 87
Figura 71 Anexo I - Criação dos Arquivos 87
Figura 72 Anexo I - Adição de Arquivos 88
Figura 73 Anexo I - Workspace 88
Figura 74 Anexo I - Modificação dos Arquivos 89
Figura 75 Anexo I - Arquivos da Pasta myProject1 Alterados 89
Figura 76 Anexo I - Somente o Arquivo do Tipo 'C' é Criado 90
Figura 77 Anexo I - Tipos de Arquivos 90
Figura 78 Utilização do Socket para PIC tipo DIP 92
Figura 79 MPLAB SIM 93
Figura 80 Menu debug para o MPLAB® SIM 94
Figura 81 Programa em execução 95
Figura 82 Fim de execução da simulação. 95
Figura 83 Breakpoints 95
Figura 84 Step Into 96
Figura 85 Watch Window 97
Figura 86 Mudança no estado dos LEDs (PORTA, bits A0-A3) de zero para um. 97
Figura 87 Estímulo Assíncrono 98
Figura 88 Estímulo para troca de estado do botão 2 representado pelo PORT RD7. 98
Figura 89 Alteração do estado dos PORTs RD7 e RA1. 99
Figura 90 Seqüência de estímulos para alterar o estado dos PORTs. 99
Figura 91 Visualizando variáveis através do Watch Window 100
Figura 92 Mensagem devido a não inicialização das variáveis 100
Figura 93 Visualização das variáveis inicializadas no watch window 101
Figura 94 Visualização da variação dos valores das variáveis no watch window 101
Figura 95 Alteração manual do valor das variáveis 101
Figura 96 Opção *Trace All* para utilização do *Logic Analyzer* 102
Figura 97 *Logic Analyzer* 102
Figura 98 Seleção dos Registradores para o *Logic Analyzer* 103
Figura 99 Visualização da alteração do valor dos PORTs com o *Logic Analyzer* 103
Figura 100 Opção Uart1 IO para visualização da Saída UART 105

Figura 101 Saída do programa de simulação da UART 105
Figura 102 Configuração do clock 106
Figura 103 Breakpoint - Stopwatch 107
Figura 104 Stopwatch 108
Figura 105 Stopwatch 2 108
Figura 106 Stopwatch 3 109
Figura 107 Stopwatch 4 109
Figura 108 Esquema Elétrico dos Botes do kit EXPLORER16 BR 111
Figura 109 Esquema Elétrico dos Leds do kit EXPLORER16BR 112
Figura 110 Circuito resultante para o PORT RA7 113

1. Introdução

A segunda etapa da disciplina de Laboratório de Sistemas, Processadores e Periféricos refere-se aos princípios básicos de projeto com microcontroladores através da programação de seus periféricos que lhes servirão como base para o desenvolvimento de Sistemas Embutidos.

Um microcontrolador (MCU) é um microcomputador em um único *chip*, contendo um processador, memória e periféricos de entrada/saída. De forma bem genérica e simples podemos considerar:

Microcontroladores = CPU + periféricos (interfaces programáveis do microcontrolador com o mundo externo)

A programação dos periféricos dos MCUs se concentram no entendimento do funcionamento do periférico que se quer usar e na configuração correta dos registradores envolvidos na programação do mesmo. Tanto a forma de funcionamento de um periférico, quanto as informações sobre os registradores envolvidos na sua programação são detalhamente encontrados no *datasheet* do MCU escolhido para o desenvolvimento da aplicação.

Ao terminar de ler as instruções a seguir você saberá:

1) O que você deverá apresentar ao professor no início de cada aula prática.
2) O que fazer nos dias das aulas de laboratório.
3) O material que deverá usar para realizar as atividades.

Essa segunda etapa da disciplina de Laboratório de Sistemas Processadores e Periféricos é composta por 7 aulas (ou 7 atividades). Cada atividade tratará de um periférico de um MCU do fabricante *Microchip*. O modelo de MCU que usaremos no curso é o PIC32MX360F512L que é constituído por uma CPU de 32 bits e se encontra montado em um soquete na placa de desenvolvimento de experimentos chamada Explorer 16 BR do fabricante *Labtools*. Tanto o *datasheet* do MCU, quanto o manual do usuário da placa Explorer 16 BR serão documentos essenciais para sua consulta no desenvolvimento das atividades propostas.

Em cada atividade proposta você encontrará as seguintes partes:

1) **Introdução**: onde o periférico do MCU lhe será apresentado e os princípios de funcionamento e programação lhes serão explicados.

2) **Registradores** usados na programação desse periférico. Nessa parte são apresentados os registradores do MCU que serão usados na programação.

3) **Atividade 1**: É uma atividade pronta (com código fonte completo) completamente funcional.

O que você deverá fazer em sua casa: **criar** um **novo** projeto com o código da atividade 1 no ambiente de desenvolvimento de projetos, compilá-lo, simulá-lo e analisar os resultados da simulação. Você aprenderá nas seções 4, 5 e 6 dessa apostila, como instalar e usar o ambiente de desenvolvimento de projetos.

O que você deverá fazer no laboratório: tirar dúvidas com o professor sobre a atividade, gravar o código da atividade 1 no kit Explorer 16 BR e testá-lo na presença do professor.

4) **Atividade 2**: É uma atividade simples que demanda uma pequena modificação baseado no código da atividade 1.

O que você deverá fazer em sua casa: **criar** um **novo** projeto com o código da atividade 2 no ambiente de desenvolvimento, compilá-lo, simulá-lo e analisar os resultados da simulação.

O que você deverá fazer no laboratório: tirar dúvidas com o professor, mostrar ao professor a simulação da atividade 2 e gravar o código da atividade 2 no kit para testá-lo na presença do professor.

5) **Atividade 3**: É uma atividade de complexidade média que exige que você desenvolva um novo código para uma situação proposta.

O que você deverá fazer em sua casa: criar um **novo** projeto e programar um código da atividade 3 no ambiente de desenvolvimento, compilá-lo, simulá-lo e observar o funcionamento do periférico na simulação.

O que você deverá fazer no laboratório: tirar dúvidas com o professor, mostrar ao professor seu código, a simulação da atividade 3 e gravar o código da atividade 3 no kit para testá-lo na presença do professor.

O material base que vocês usarão para realização dessas atividades são classificados em material de Software, Hardware e Documentação como mostro a seguir:

Software:
1) O ambiente de desenvolvimento e Simulação de projetos usando MCUs da Microchip, chamado MPLAB.
Baixe-o gratuitamente em:
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002
2) O compilador de programas em linguagem C escrito para MCUs de 32 bits da Microchip, chamado de C32.
Baixe a versão acadêmica gratuitamente em:
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en536656

Hardware:
1) O MCU PIC32MX360F512L
2) O Kit Explorer 16 BR: kit com o MCU e componentes eletrônicos (leds, sensores, displays, etc.) e mecânicos (botões ou teclado, conectores, etc.) para testes das práticas.
3) O gravador ICD2 BR: necessário para gravar seus programas compilados e simulados no MCU instalado no kit.

Documentação:
1) Apostila_PIC_UFMG.pdf - Nessa apostila você encontrará informações sobre como instalar os *softwares*, criar projetos no ambiente de desenvolvimento, compilar seus projetos e simulá-los. Nessa apostila você também encontrará as 7 atividades propostas com as 3 atividades por atividade que foram mencionadas anteriormente.
2) O manual do usuário do kit Explorer 16 BR - Nesse manual você poderá observar onde cada componente (eletrônico e mecânico) da placa está ligado fisicamente a qual pino do MCU.
3) O *datasheet* do MCU PIC32MX360F512L - Nesse *datasheet* você encontrará informações sobre pinagem, periféricos disponíveis e seu funcionamento, registradores usados na programação desses periféricos, pinos que um determinado periférico usa, etc.

Todos os documentos citados se encontram no ambiente *Moodle* disponível para *download*, ou então no próprio site dos fabricantes mencionados.

Esta apostila tem por objetivo conduzir o aluno a explorar o uso de periféricos e funcionalidades do microcontrolador PIC32MX360F512L, bem como disponibilizar informações relevantes sobre o uso do kit de desenvolvimento EXPLORER 16 BR da (*Labtools*) para testes das atividades práticas propostas. Para essa etapa da disciplina de Laboratório de Sistemas, Processadores e Periféricos é indispensável que o aluno possua conhecimento em linguagem C.

Nesta apostila ainda serão abordados os seguintes tópicos:

- Diferenças entre microcontroladores e microprocessadores.
- Apresentação do ambiente de desenvolvimento de programação, simulação e gravação, de microcontroladores do fabricante Microchip conhecido por MPLAB®
- Apresentação do compilador C, chamado de MPLAB® - C32
- Apresentação da Placa de gravação de microcontroladores, conhecida por ICD2 BR,
- A criação de projetos no MPLAB®
- Apresentação do ambiente de simulação do MPLAB® conhecido por MPLAB - SIM.
- As atividades práticas propostas.

As atividades práticas abordarão os seguintes periféricos:

1. Configuração de Interface que controla os pinos do MCU para Leitura de botões e acionamento de LEDs.
2. Configuração da Interface de Tratamento de Interrupções de Eventos Externos.
3. Configuração de Temporizadores (*Timers*) e Contadores de eventos externos (*Counters*).
4. Configuração de Interface para Comunicação com Displays de Cristal Líquido (*LCDs*) alfa-numéricos.
5. Configuração de Conversores Analógicos/Digitais (Conversores A/D).
6. Configuração de Interface para Comunicação serial UART com RS-232.
7. Configuração do Periférico de Modulação de Largura de Pulsos (*PWM*).

Concluímos reforçando que cada aula prática será composta por três atividades. A atividade 1 é uma atividade simples e completamente funcional, na qual o aluno poderá atestar o uso e o funcionamento do periférico. Na atividade 2 ele deverá ser capaz de realizar uma modificação na atividade anterior. A terceira atividade exigirá que o aluno desenvolva um código novo.

2. Microcontroladores, microprocessadores e sistemas embarcados

O microprocessador ou CPU (*Central Processing Unit*) é um circuito integrado responsável por processar dados e executar instruções. Ele é composto basicamente por: *Caminho de dados* (ULA + Registradores + Multiplexadores) responsável por executar ou processar as instruções e *Unidade de controle,* responsável por gerenciar o tráfego de informação nos barramentos.

Os microprocessadores não possuem periféricos tais como interfaces programáveis de entradas e saídas de dados, conversores Analógicos/Digitais (A/D), Temporizadores *(timers)*, Módulos para comunicação e transmissão de dados programáveis e Módulos PWM (Modulação de Largura de Pulsos – *Pulse Width Modulation*) integrados.

O microcontrolador por sua vez é constituído por uma CPU mais simples e uma diversidade de periféricos programáveis em um único *chip*.

Mesmo não possuindo os periféricos integrados em um único *chip* os microprocessadores são amplamente utilizados em aplicações que demandam processamento de dados intenso e diversificado, haja visto que sua capacidade de processamento de dados é muito superior a dos microcontroladores.

É importante compreender que existe no mercado um grande número de fabricantes de microcontroladores tais como: (Microchip), (Texas Instruments), (Atmel), dentre outras.

Os microcontroladores são muito usados no projeto de Sistemas Embutidos (ou Sistemas Embarcados).

Um Sistema Embarcado (ou Sistema Embutido) é um sistema microprocessado no qual a unidade de processamento (CPU) é completamente dedicada ao dispositivo ou sistema que ele controla. Diferente de computadores de propósito geral, como um computador pessoal ou *notebook*, um sistema embarcado realiza um conjunto de tarefas predefinidas, geralmente com requisitos bem especificados.

Desktops e *notebooks* possuem processadores (Ex.: Intel, Pentium, Athlon, etc.) e periféricos (monitor, teclado, mouse, impressora, etc) que podem ser utilizados para interface com outros sistemas ou com o ser humano (Ex.: edição de texto, acesso a Internet, Banco de dados, entre outros), ou seja, os computadores não são concebidos para uma finalidade específica como sistemas embutidos.

Os sistemas embarcados por sua vez possuem um ou mais microcontroladores e executam tarefas especificas conforme foram projetados segundo seus requisitos de projeto. Como exemplo de sistema embarcado podemos citar o controle de acesso de uma catraca, celulares, relógios digitais, porta retratos digitais, *smart cards*, brinquedos, etc.

O objetivo principal deste curso é fornecer a base para o projeto com microcontroladores de forma que o aluno possa construir sistemas embarcados variados tais como : Letreiros luminosos, controladores, sistemas de acionamento de máquinas, aplicações de automação e domótica, etc. No capitulo 3 e 4 serão apresentados o material e o ambiente de desenvolvimento necessário para a realização de projetos com microcontroladores que serão usadas no nosso curso.

3. Material utilizado no curso

O kit de desenvolvimento da (*Labtools*) contém os seguintes materiais:

- Uma placa de desenvolvimento EXPLORER 16 BR.

Figura 1 Explorer 16 BR

- Uma placa de gravação ICD2 BR.

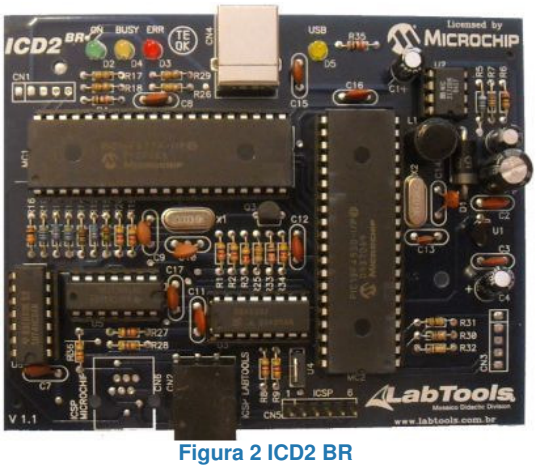

Figura 2 ICD2 BR

- Um *Plugin* montado com o microcontrolador PIC32MX360F512L

Figura 3 Plugin PIC32MX360F512L

- Um conector RJ12

Figura 4 Conector RJ12

- Um cabo USB

Figura 5 Conector USB

- Uma fonte de alimentação

Figura 6 Fonte Alimentação

4. MPLAB® IDE e MPLAB® C32

O (Mplab IDE) (*Integrated Development Environment*) é um ambiente de trabalho para programação e simulação de projetos baseados em PIC. A Microchip® fornece ainda um compilador em linguagem C para PIC. O (Mplab C18) é o compilador da Microchip® para a família PIC 8 bits, o (Mplab C30 2010) é o compilador para a família de 16 bits: PIC24, dsPIC30F e dsPIC33F e o (Mplab C32 2010) para a família 32 bits, como o PIC32MX360F512L que iremos utilizar durante o curso.

É importante que o aluno tenha consciência de que, da mesma forma que existe uma infinidade de microcontroladores no mercado (PIC, dsPIC, DSP, ARM, etc) temos, além do MPLAB®, diversas plataformas (IDE) e compiladores para desenvolvimento de projetos, por exemplo: (HI-TECH 2010), (MicroC 2010), (CCS 2010), dentre outros, que apresentam particularidades próprias para descrição de código fonte e portanto não são compatíveis uns com os outros.

4.1. Instalação MPLAB® IDE

Para instalar o MPLAB® IDE insira o CD Explorer16 BR fornecido com o kit (Figura 7), ou então baixe uma versão mais atualizada do MPLAB® IDE do *site* da *Microchip* (http://www.microchip.com) conforme o Sistema Operacional instalado em seu computador.

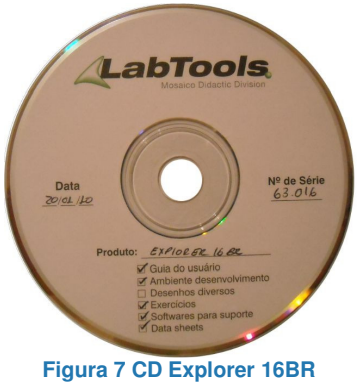

Figura 7 CD Explorer 16BR

No CD você encontrará as pastas mostradas na Figura 8 a seguir.

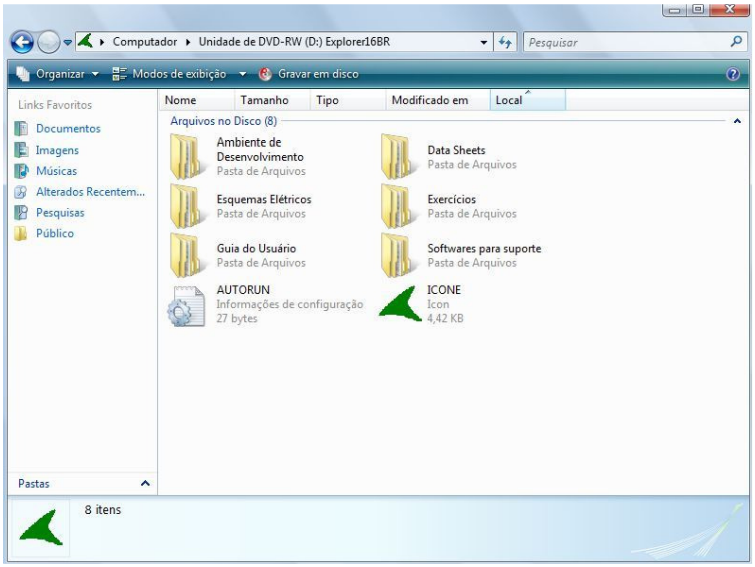

Figura 8 Pastas CD Explorer16BR

Por instante iremos nos preocupar somente com a pasta *"Ambiente de Desenvolvimento"*. Vá em: ***"Ambiente de Desenvolvimento >> MPLAB >> MPLAB_8.40 >> Setup"***.

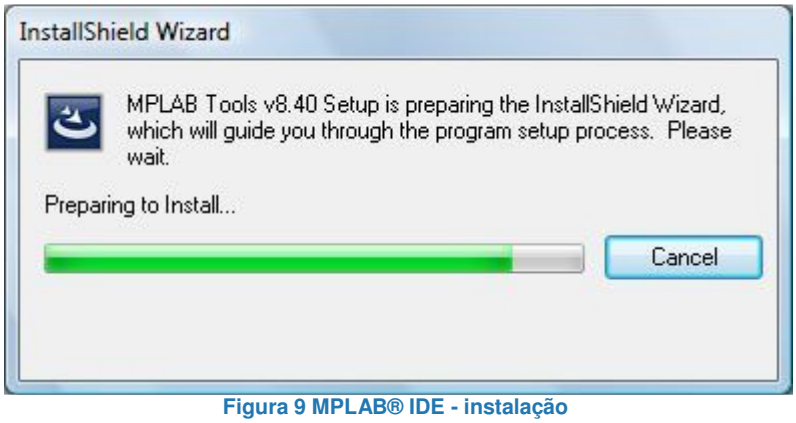

Figura 9 MPLAB® IDE - instalação

Na janela que se abriu clique ***next***, leia o termo de compromisso e clique em ***"I accept the terms of the license agreement"*** e novamente em ***next***. (Figura 10)

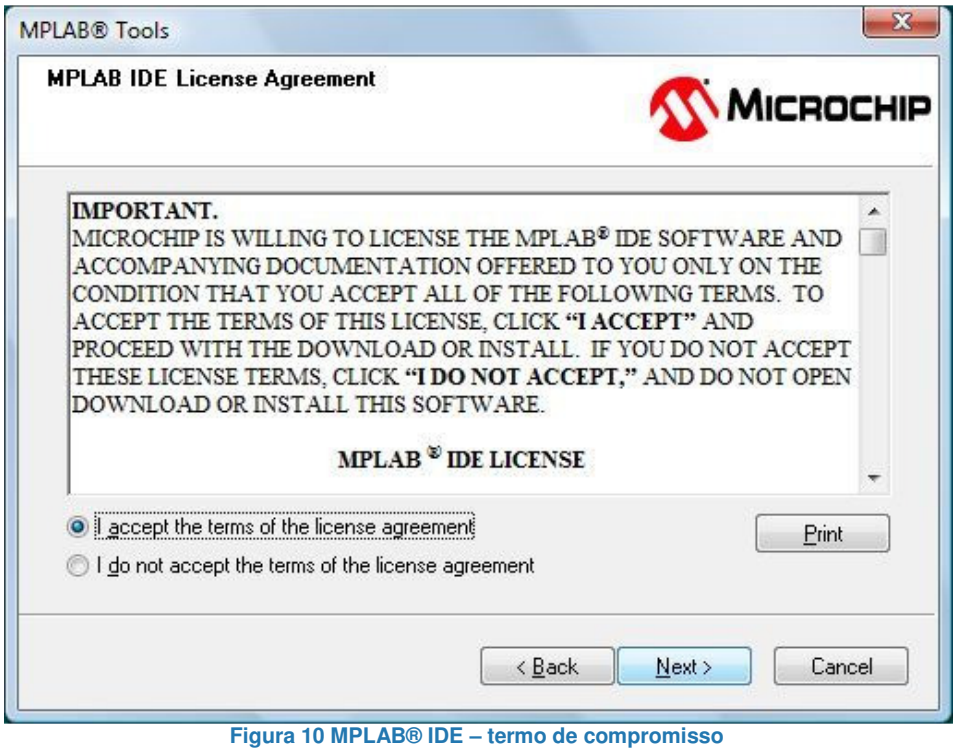

Figura 10 MPLAB® IDE – termo de compromisso

Selecione o modo de instalação *"Complete"* para instalar todas as ferramentas ou o modo *"Custom"* para selecionar aqueles de seu interesse caso você seja um usuário avançado.

No nosso caso iremos fazer a instalação ***"Complete"***. (Figura 11)

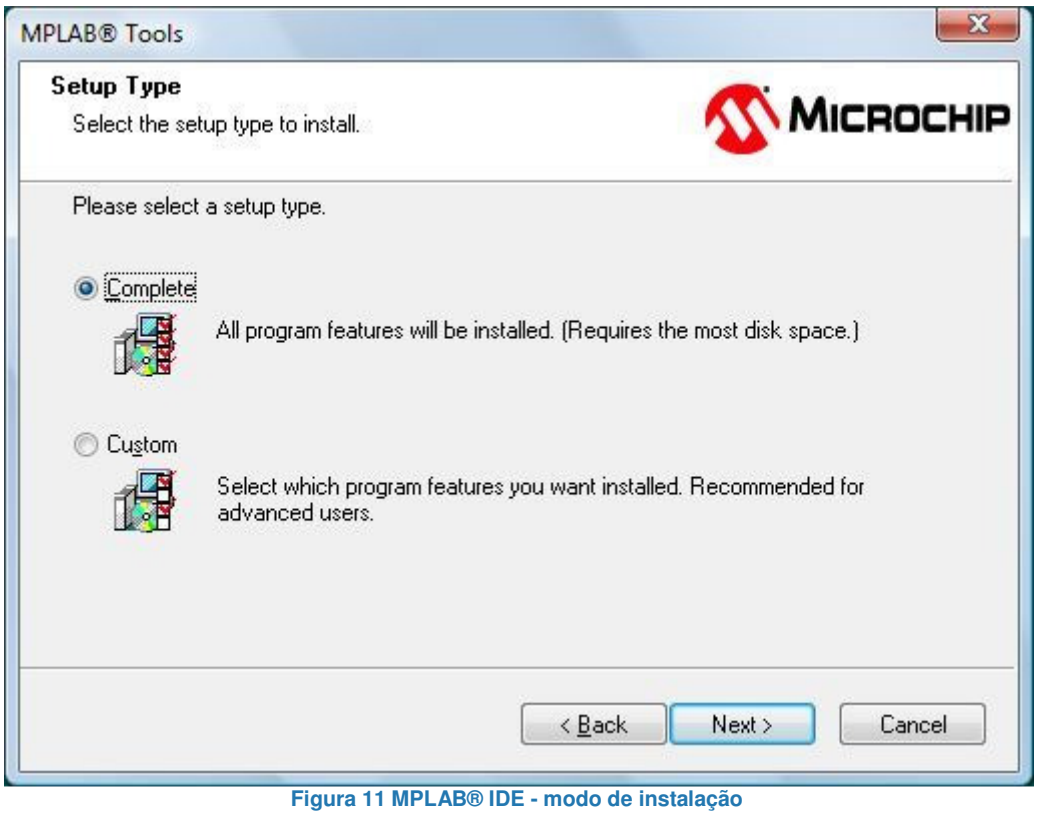

Figura 11 MPLAB® IDE - modo de instalação

Para que não ocorra problemas futuros quando precisarmos referenciar a pasta de instalação do MPLAB® deixe-a como está: ***"c:\Program Files\Microchip\"***. (Figura 12)

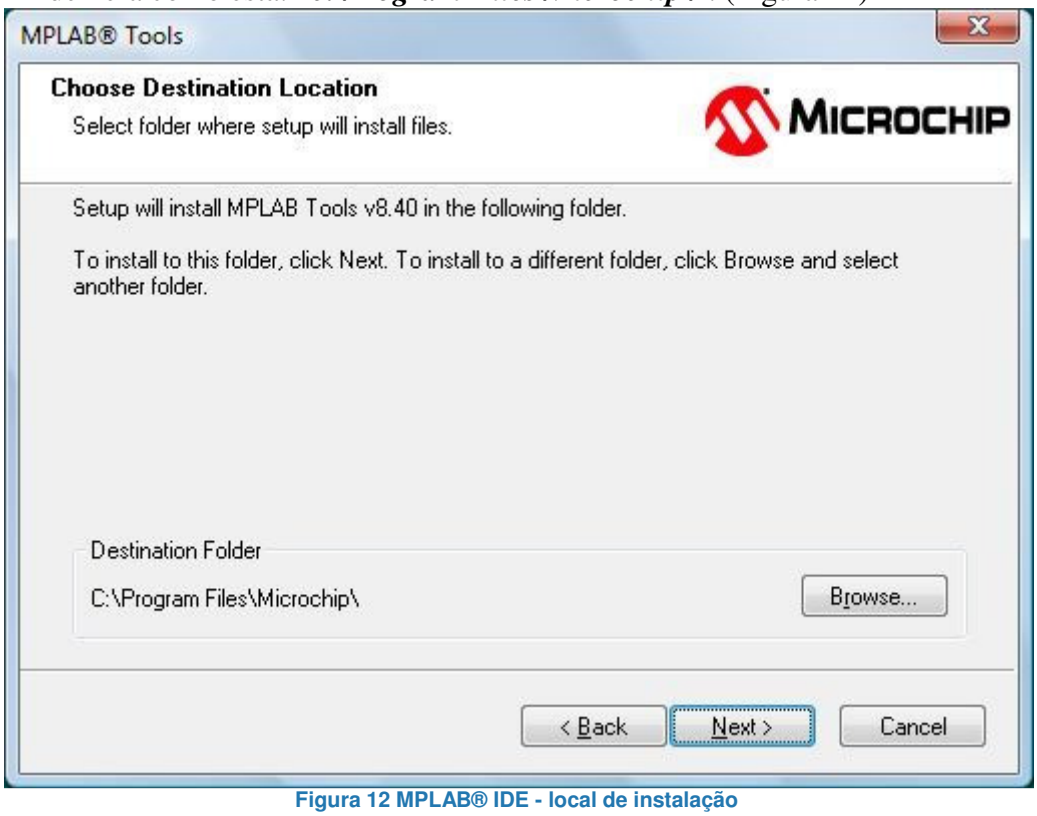

Figura 12 MPLAB® IDE - local de instalação

Leia o termo de compromisso de utilização do software, clique em ***"I accept the terms of the license agreement"*** e posteriormente em ***next***. (Figura 13)

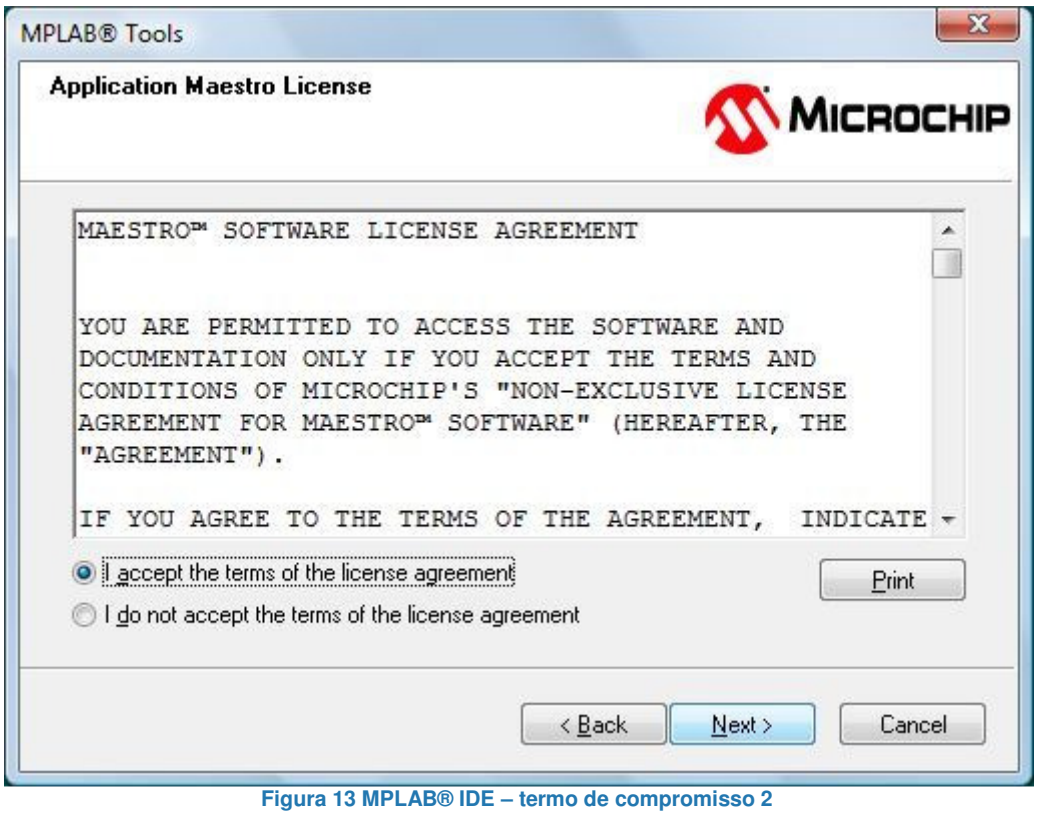

Figura 13 MPLAB® IDE – termo de compromisso 2

Leia o termo de compromisso de utilização do compilador C32, clique em ***"I accept the terms of the license agreement"*** e posteriormente em ***next***. (Figura 14)

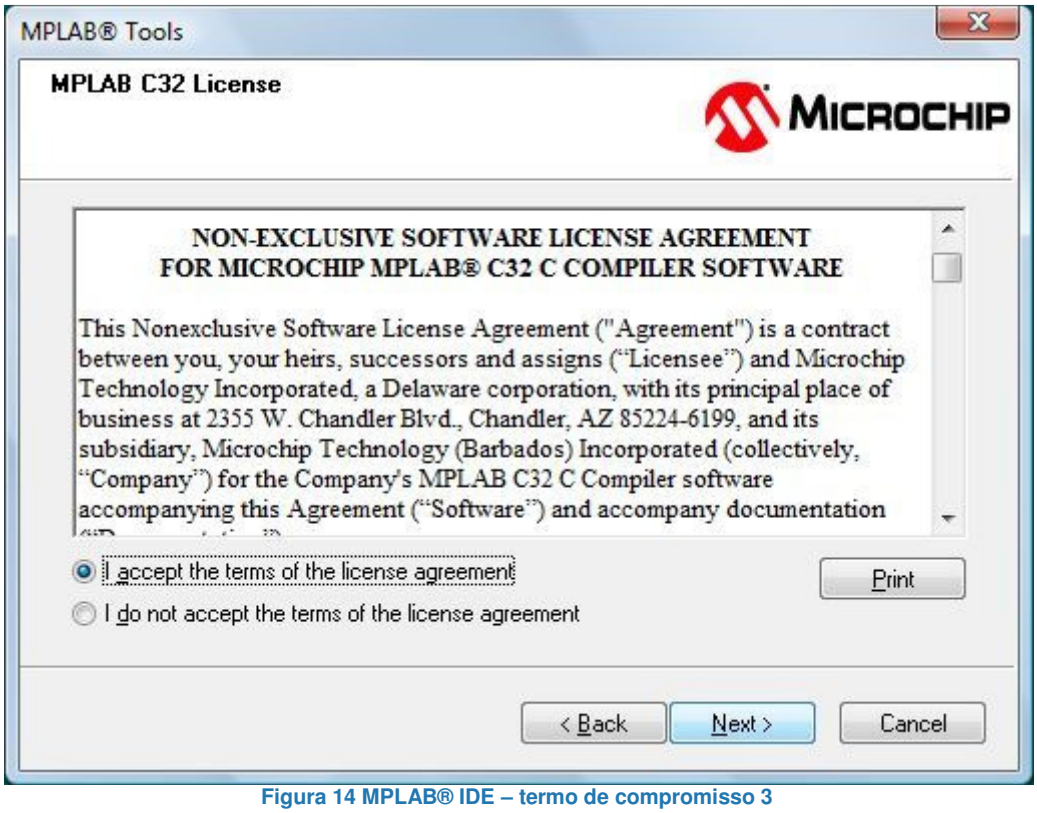

Figura 14 MPLAB® IDE – termo de compromisso 3

Revise se o diretório bem como os componentes a serem instalados estão corretos. Caso esteja tudo ok clique em ***next*** para iniciar a instalação. (Figura 15)

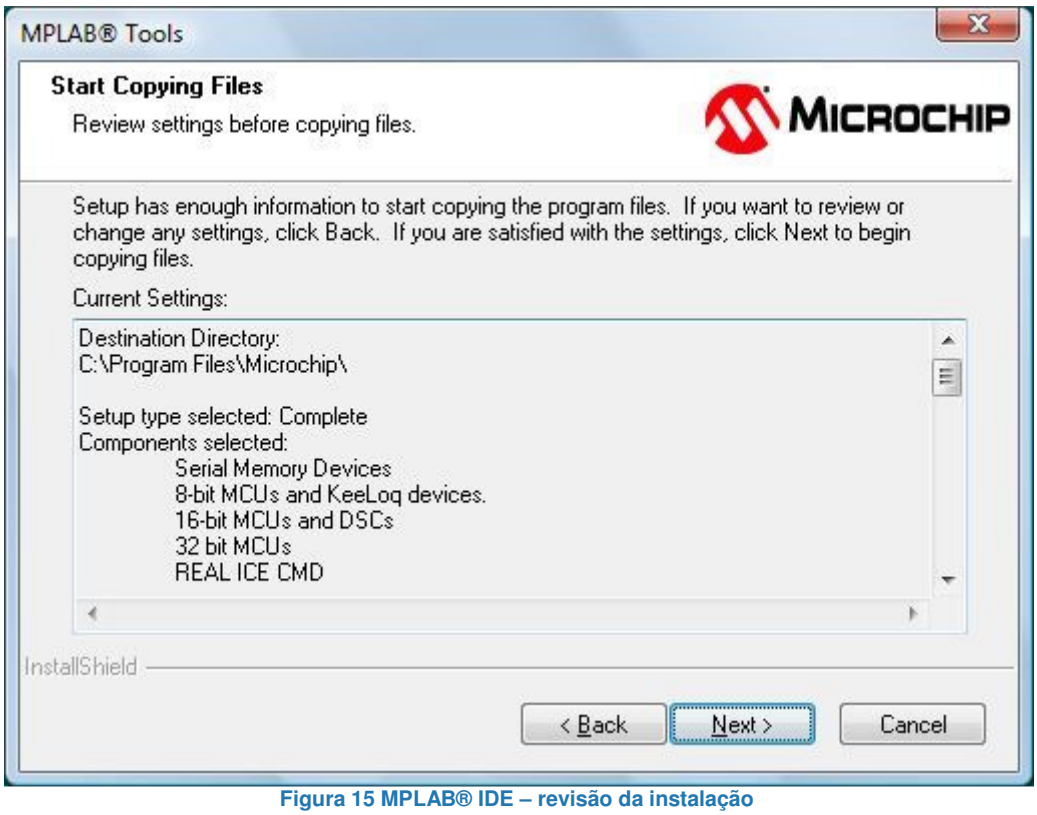

Figura 15 MPLAB® IDE – revisão da instalação

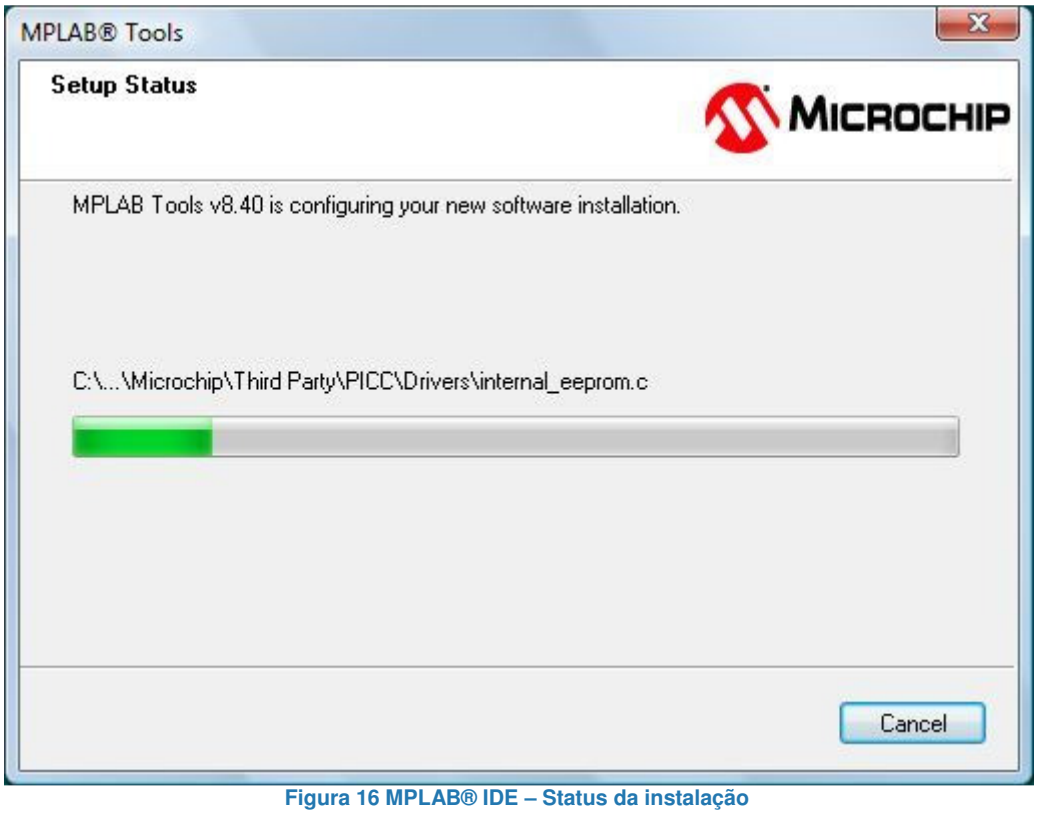

Figura 16 MPLAB® IDE – Status da instalação

No final da instalação aparecerá uma mensagem (Figura 17) solicitando se você deseja instalar o compilador HI-TECH C, ***clique em não*** haja visto que iremos utilizar o MPLAB® C32.

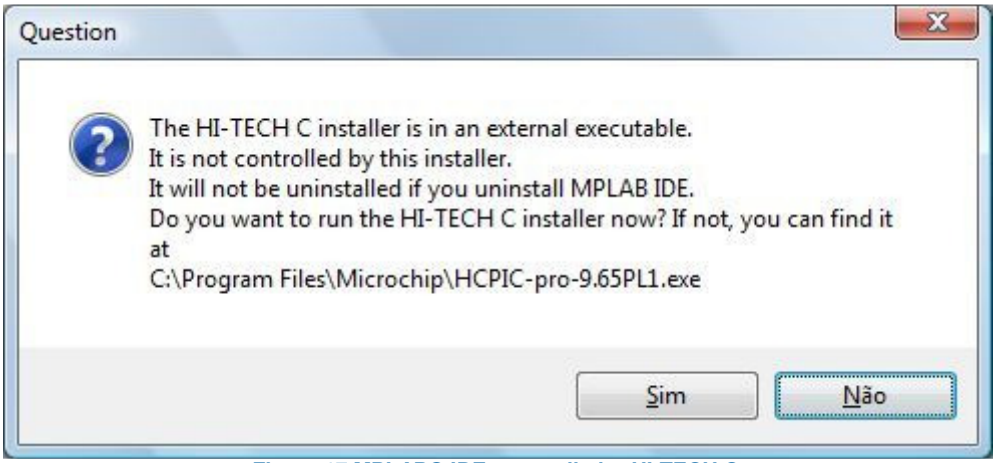

Figura 17 MPLAB® IDE – compilador HI-TECH C

Reinicie o computador para atualizar as configurações. (Figura 18)

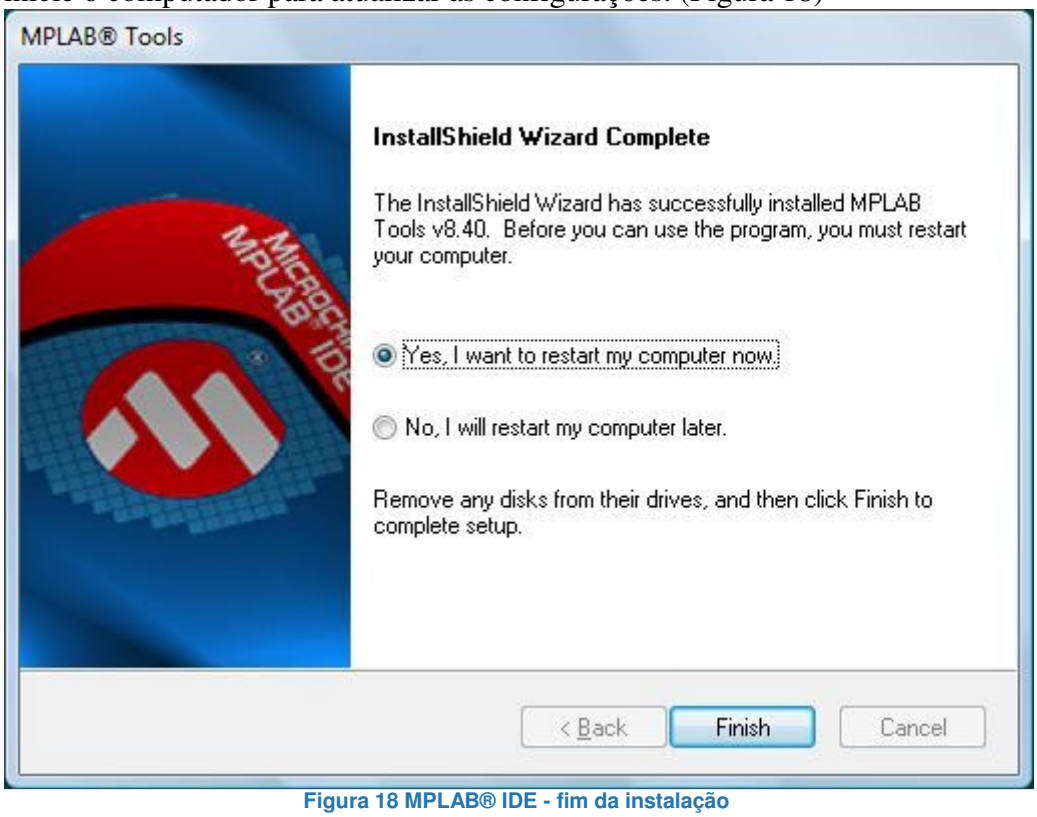

Figura 18 MPLAB® IDE - fim da instalação

Pronto, com o MPLAB® IDE instalado você poderá acessá-lo clicando no ícone que se encontra na área de trabalho ou indo até o diretório no qual ele foi instalado: ***"C:\Program Files\Microchip\MPLAB IDE\Core\MPLAB.exe"***. (Figura 19)

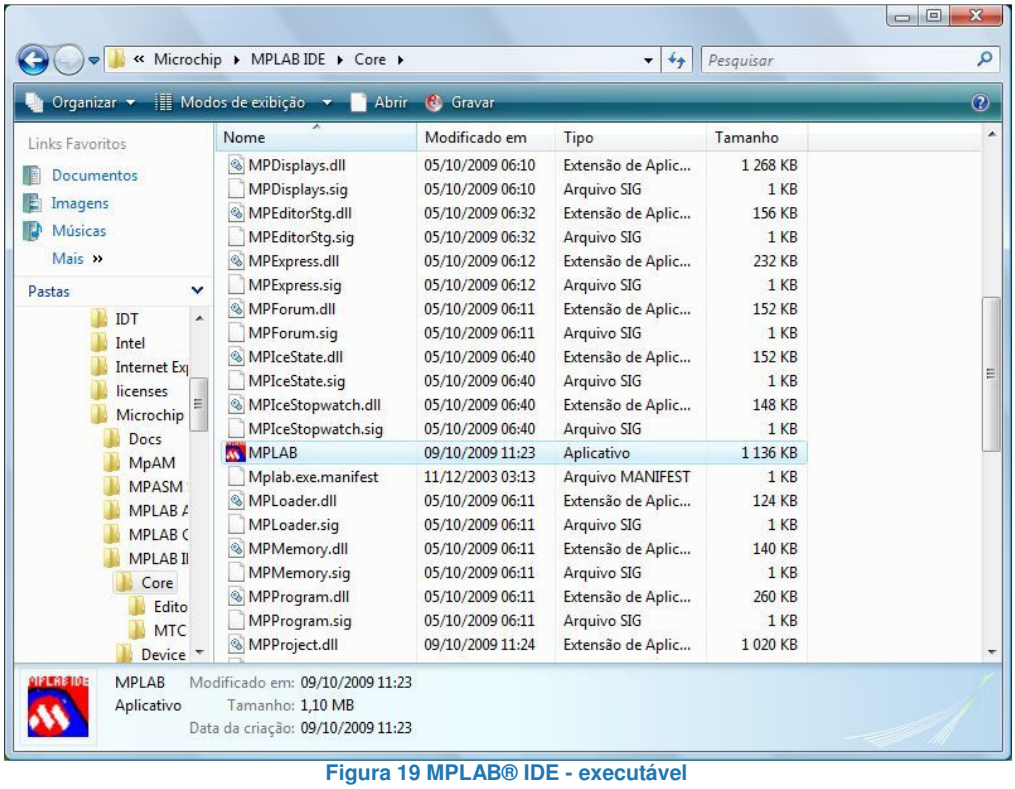

Figura 19 MPLAB® IDE - executável

4.2. Instalação MPLAB® C32

Para instalar o compilador C para PICs de 32 bits insira o CD Explorer16BR fornecido com o kit. Vá em: “Ambiente de Desenvolvimento >> C32 >> MPLAB-C32 -Academic-v105”.

Clique em next na tela de boas vindas. (Figura 20)

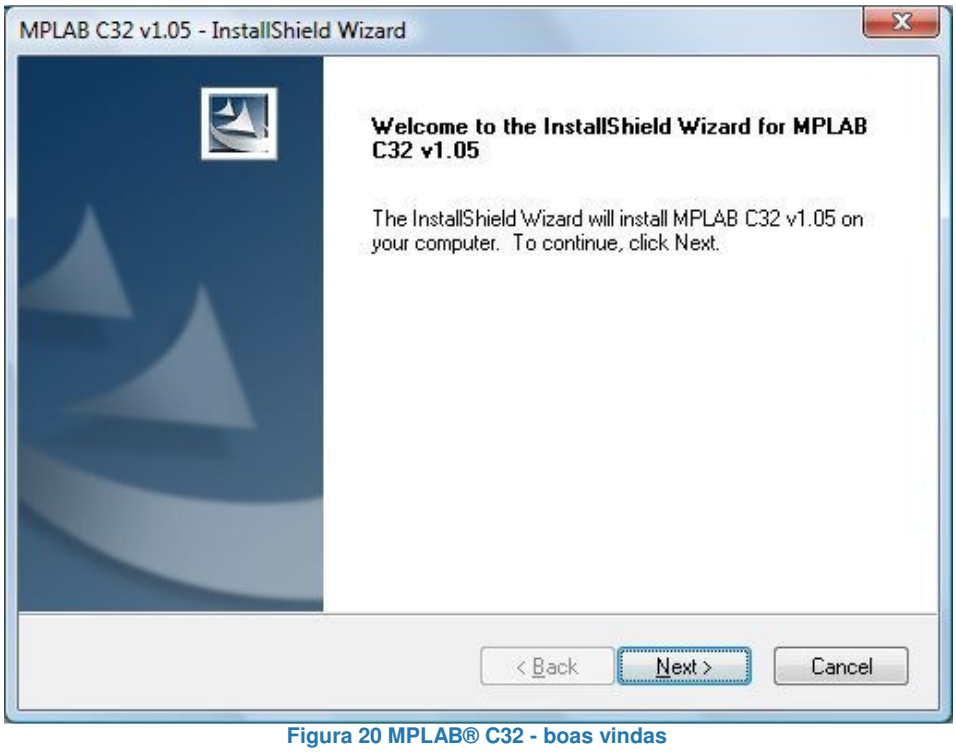

Figura 20 MPLAB® C32 - boas vindas

Leia o termo de compromisso, clique em ***“I accept the terms of the license agreement”*** e posteriormente em ***next***. (Figura 21)

Figura 21 MPLAB® C32 - termo de compromisso

Para que não ocorra problemas futuros quando precisarmos referenciar a pasta de instalação do MPLABC32 deixe-a como está: ***"C:\Program Files\Microchip\MPLAB C32"***. (Figura 22)

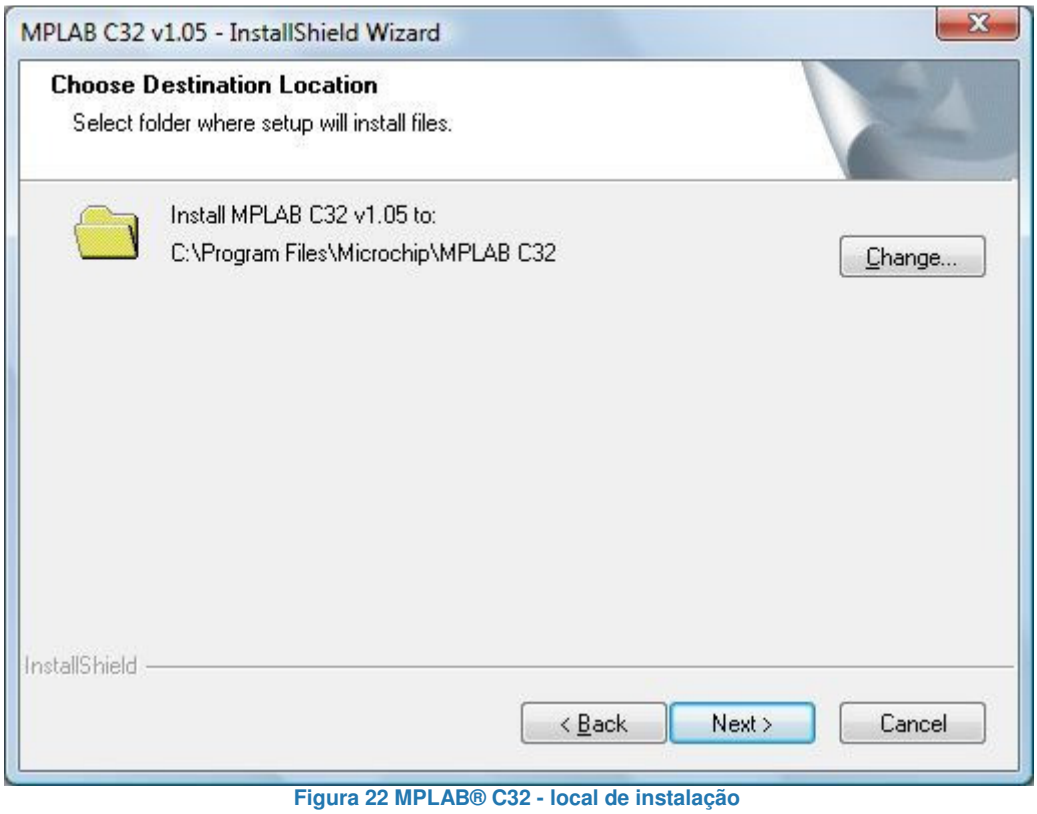

Figura 22 MPLAB® C32 - local de instalação

Clique ***"install"*** para iniciar a instalação. (Figura 23)

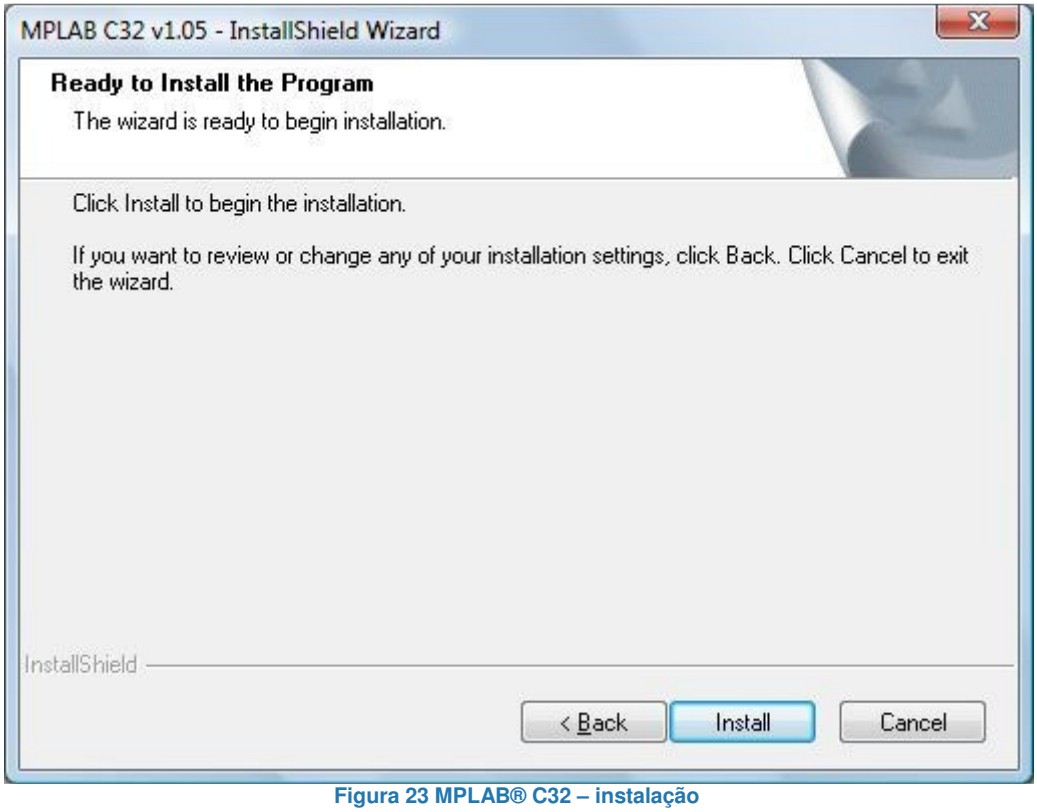

Figura 23 MPLAB® C32 – instalação

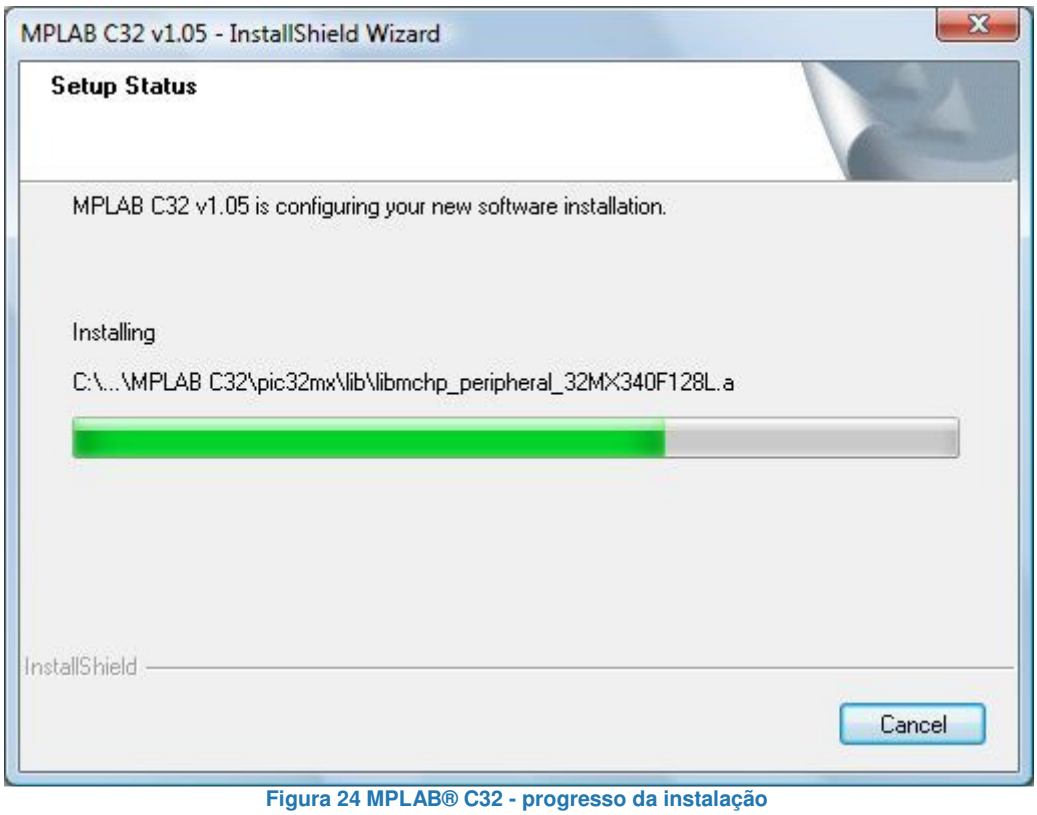

Figura 24 MPLAB® C32 - progresso da instalação

Clique em ***"finish"*** para terminar a instalação. (Figura 29)

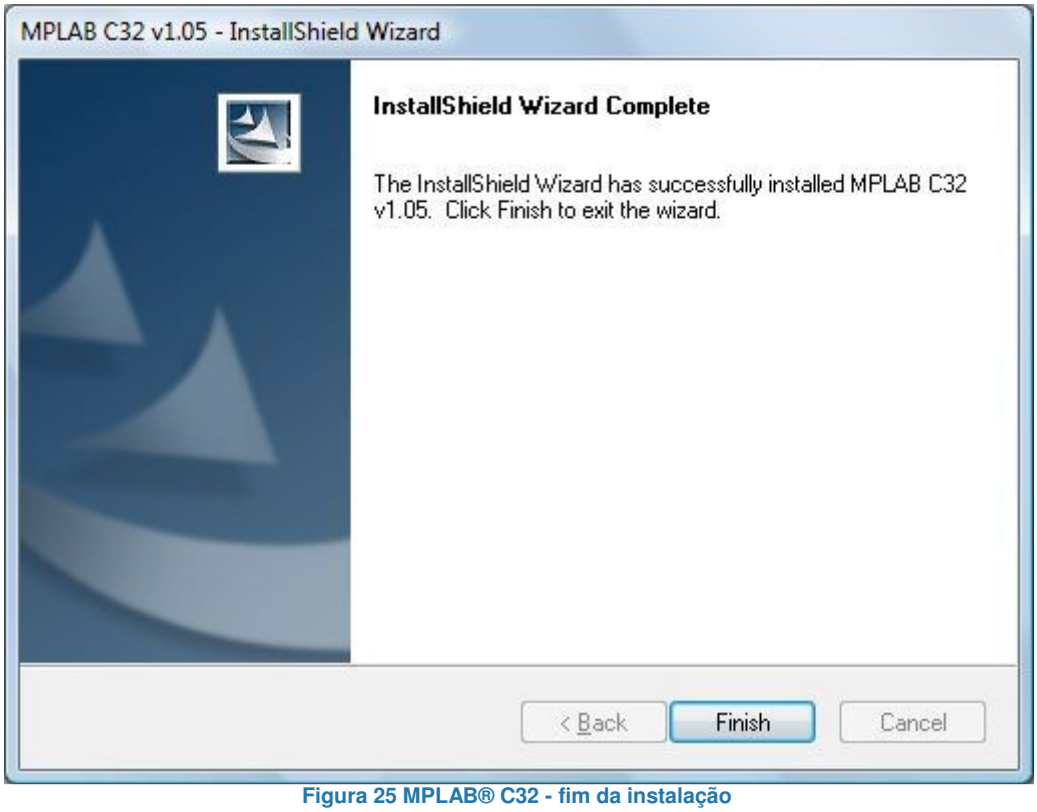

Figura 25 MPLAB® C32 - fim da instalação

5. Criação de Projeto

Para criar um projeto abra o ambiente de trabalho MPLAB®. (Figura 26)

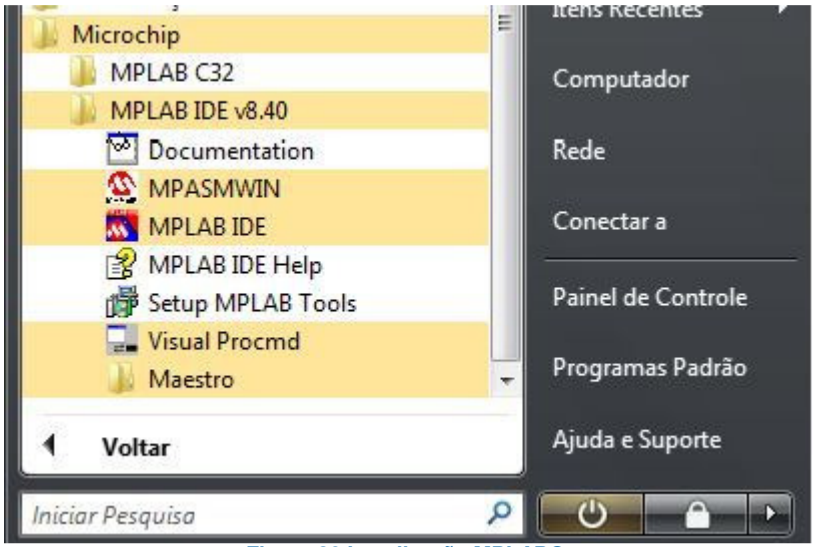

Figura 26 Localização MPLAB®

A seguinte tela deverá aparecer. (Figura 27)

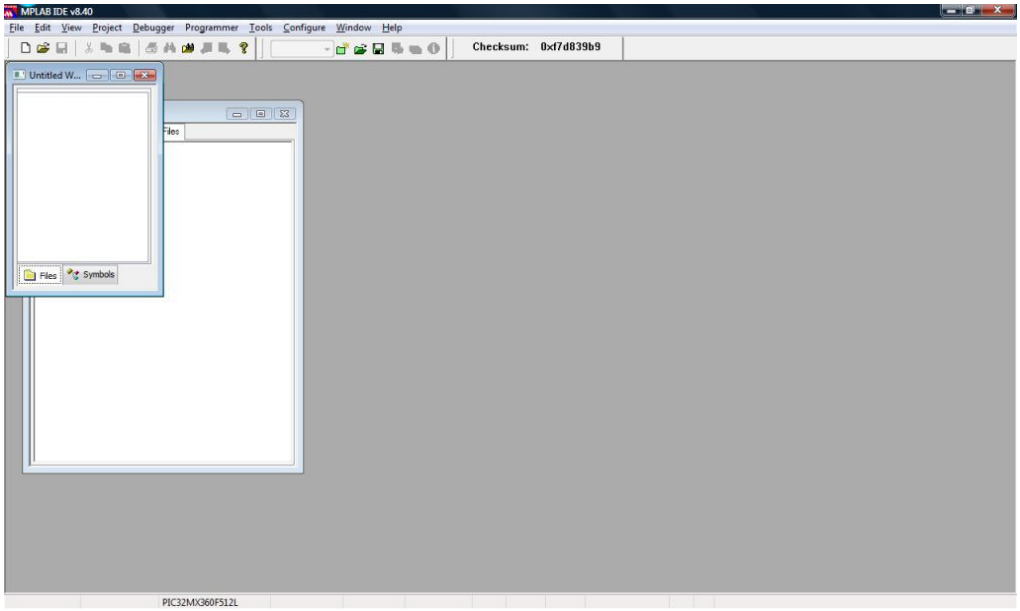

Figura 27 MPLAB® IDE

Na janela "*untitled workspace*" aparecerão todos os arquivos contidos no seu projeto e a janela "*output*" mostra o resultado da compilação.

Para criar um novo projeto faça: ***"Project >> Project Wizard..."***.

A janela que se abre é uma tela de boas vindas. Clique ***"Avançar"***. (Figura 28)

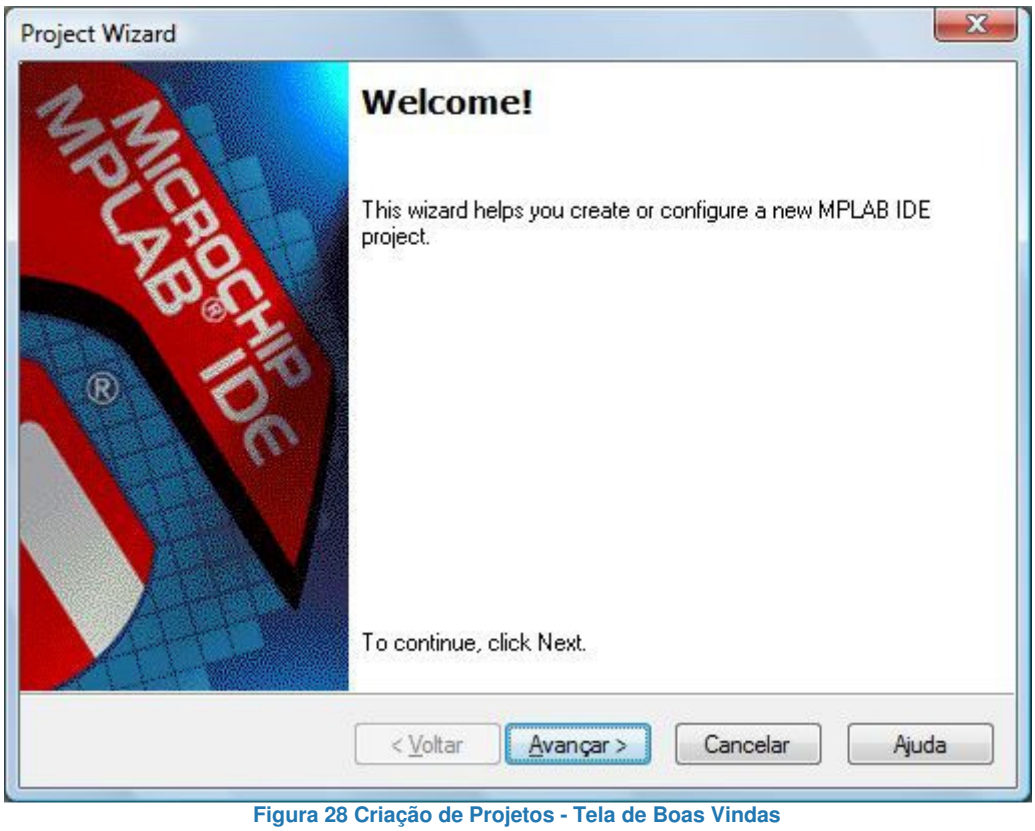

Figura 28 Criação de Projetos - Tela de Boas Vindas

Selecione o dispositivo a ser utilizado. No nosso caso selecione ***"PIC32MX360F512L"*** e clique ***"Avançar"***. (Figura 29)

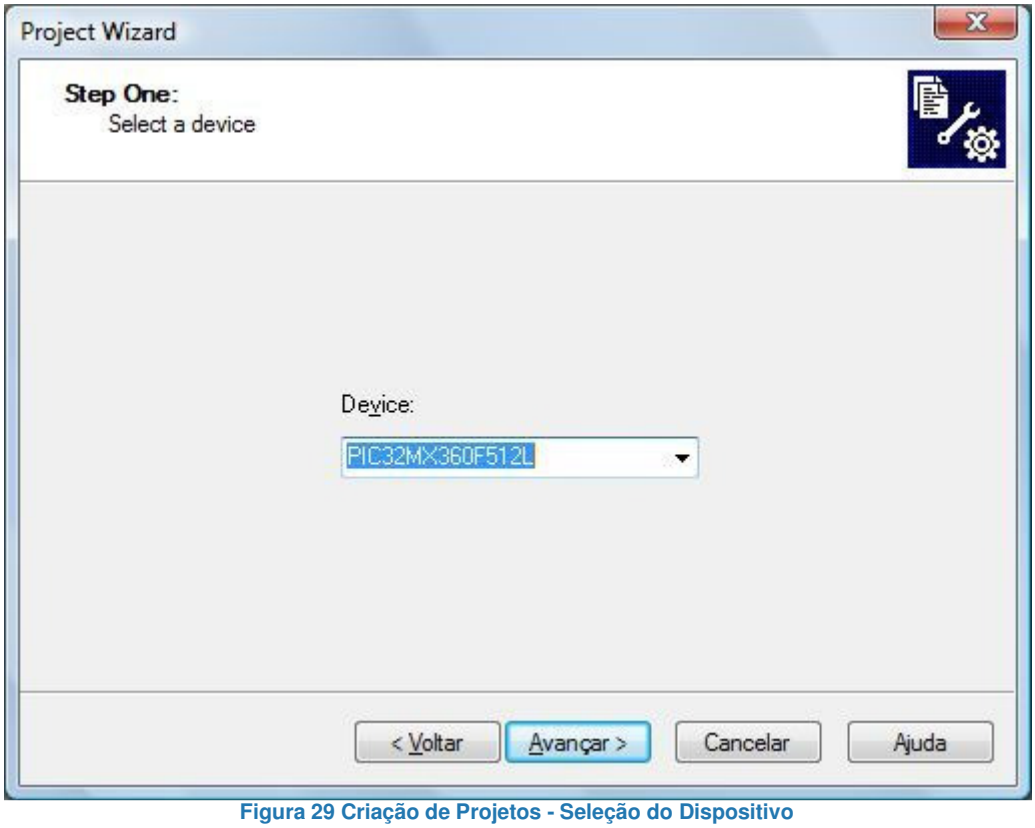

Figura 29 Criação de Projetos - Seleção do Dispositivo

Na aba ***"Active Toolsuite"*** selecione o compilador a ser utilizado. No nosso caso o ***"Microchip PIC32 C-Compiler Toolsuite"***. Clique ***"Avançar"***. (Figura 30)

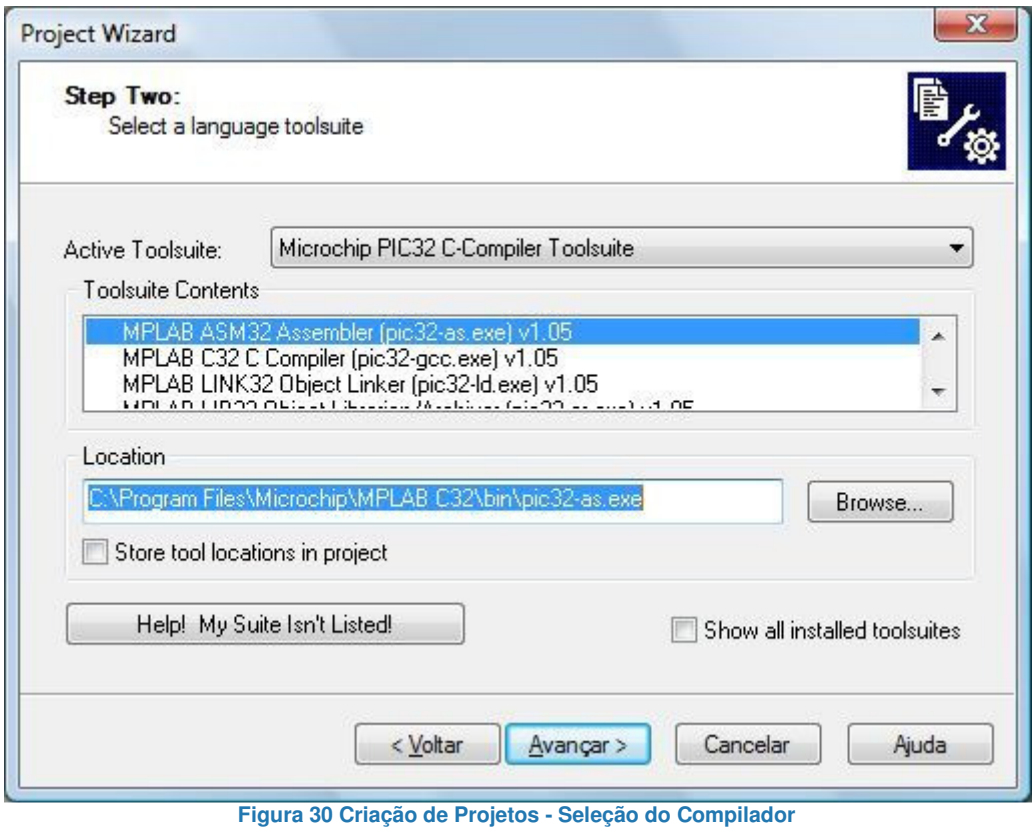

Figura 30 Criação de Projetos - Seleção do Compilador

Escolha o diretório, dê um nome para seu projeto[1] (ex: ***"c:\...\myProject"***.) e clique em ***"Avançar"***. (Figura 31)

A extensão do projeto é do tipo **".mcp"** que significa ***Microchip® Project***.

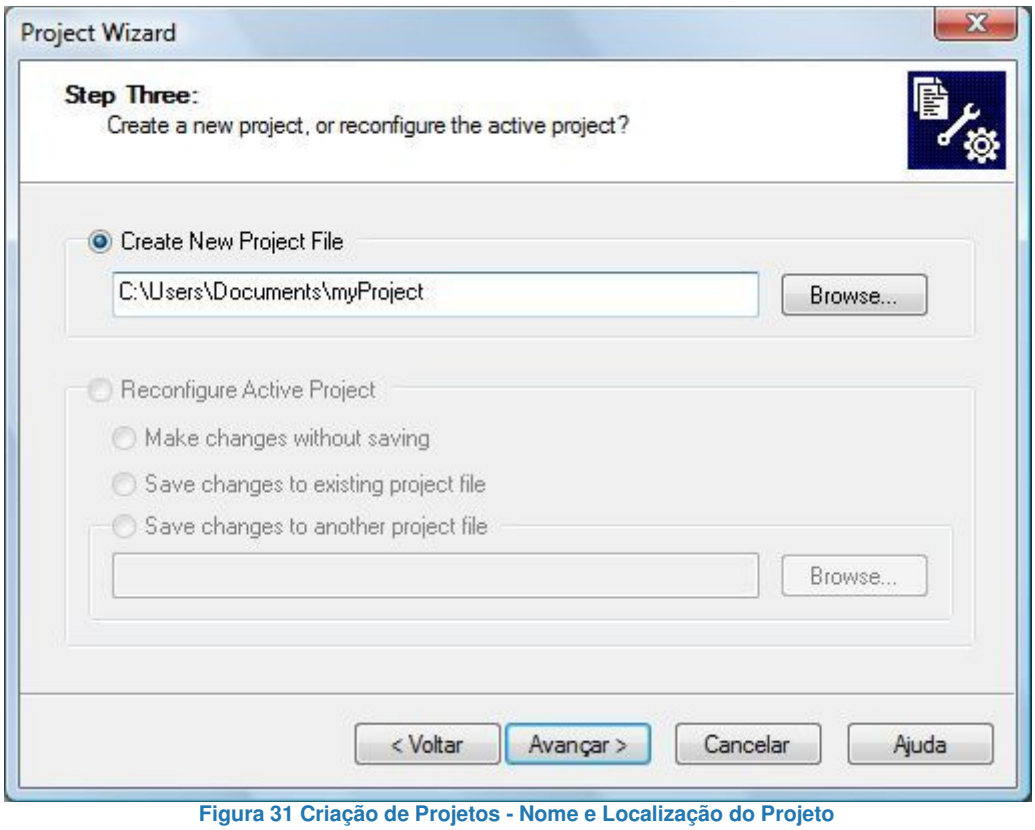

Figura 31 Criação de Projetos - Nome e Localização do Projeto

[1] Atenção : Não crie pastas com nomes separados por espaço, por exemplo, **"c:\meu projeto\myProject"**, pois seu programa poderá não compilar. Esta dica é valida para uma grande variedade de compiladores. Além do MPLAB® podemos citar o *Code Composer* da *Texas Instruments*.

Na janela que se abre você poderá inserir arquivos em seu projeto[2]. Neste instante não iremos adicionar nenhum arquivo. Clique **"*Avançar*"**. (Figura 32)

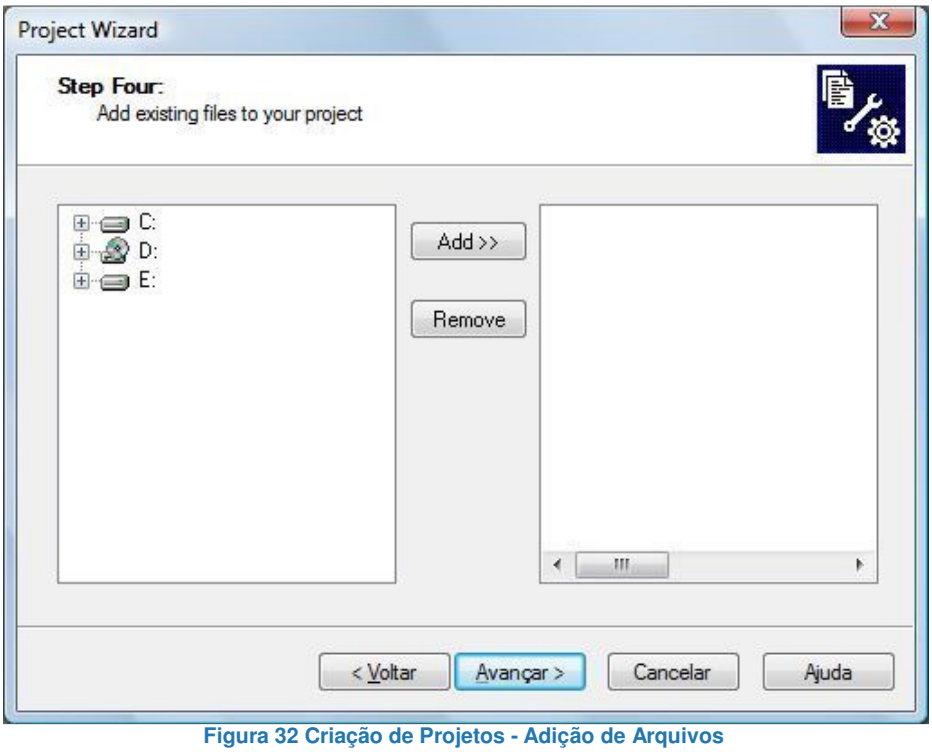

Figura 32 Criação de Projetos - Adição de Arquivos

A última tela resume as características do projeto a ser criado: dispositivo, compilador e localização. Clique **"*Concluir*"**. (Figura 33)

[2] Posteriormente o aluno poderá verificar no Anexo I : Adicionando arquivos durante a criação do projeto as 4 diferentes maneiras de se adicionar um arquivo na criação do projeto. O **Erro! Fonte de referência não encontrada.** aborda a adição de arquivos depois que o projeto já foi criado.

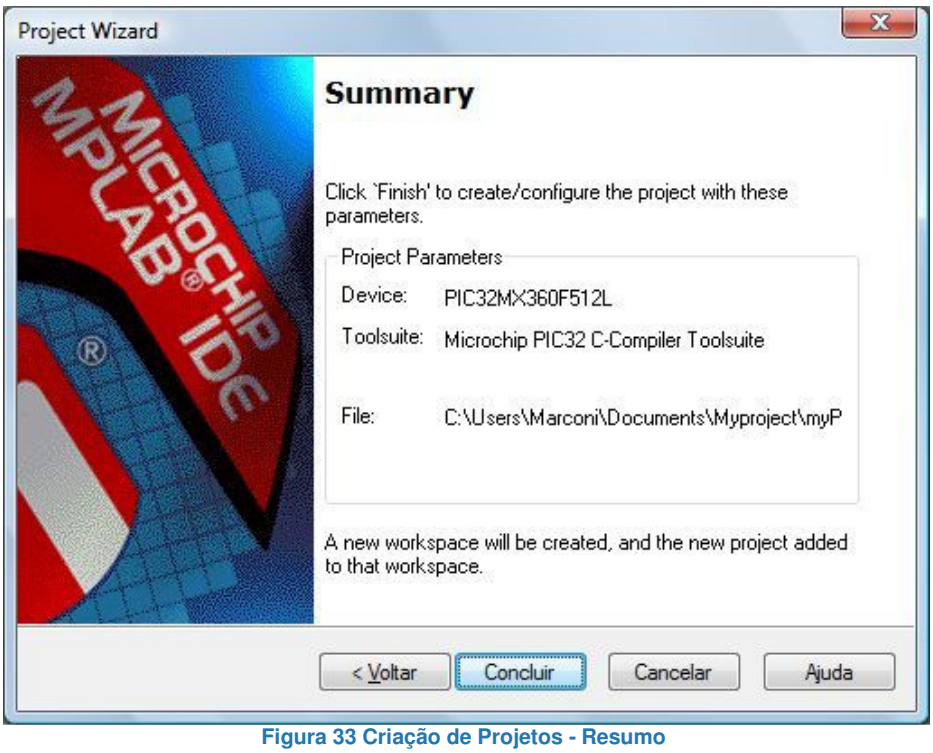

Figura 33 Criação de Projetos - Resumo

Pronto! Seu projeto está criado.

Na nova área de trabalho do MPLAB® você verá uma janela com o nome do seu projeto mas com a extensão **.mcw** (*Microchip® Workspace*), por exemplo "myProject.mcw". Esta janela mostra todos os arquivos utilizados no seu projeto. Como ainda não adicionamos nenhum arquivo ela deve estar vazia. (Figura 34)

A diferença entre .mcp e .mcw é que o *project* é o projeto em si, é ele que contém todas as informações na hora da compilação. O *workspace* é uma área de trabalho que carrega seu projeto e a disposição das janelas da maneira que você salvou pela ultima vez. Pode-se dizer, a grosso modo, que o *workspace* é o *layout* do projeto.

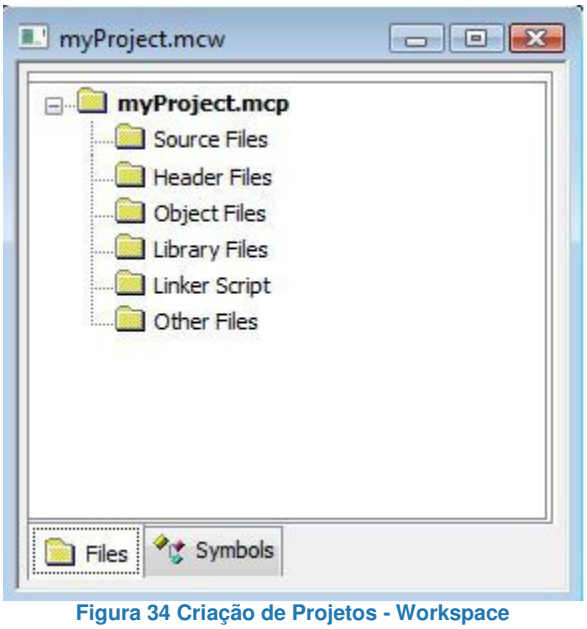

Figura 34 Criação de Projetos - Workspace

6. Gravação do Projeto no PIC

Para executar a gravação/depuração do PIC32MX360F512L coloque-o na placa EXPLORER16 BR[3] que por sua vez deve ser conectada, através do conector RJ12 (Figura 4), à placa *in circuit debugger* (ICD2BR), **Erro! Fonte de referência não encontrada.**.

Existem duas formas de utilizar a ICD2 BR. A primeira, que será a forma adotada neste curso, é conectando-a diretamente na placa de desenvolvimento EXPLORER16 BR (**Erro! Fonte e referência não encontrada.**) com o PIC colocado na mesma. A segunda maneira é colocar o PIC no *socket* que será conectado a placa ICD2 BR. Vale ressaltar que o *socket* é utilizado somente para PIC do tipo DIP (*Dual In Line Package*)[4].

A instalação do *driver* da placa ICD2 BR é feita conectando-se a mesma ao computador através da porta USB. Caso a versão do MPLAB® seja igual ou superior a 7.31 o *driver* do ICD2 BR pode ser pré-instalado durante a instalação do MPLAB®. Caso você tenha seguido os passos indicados no capitulo 4 basta conectar a placa ICD2BR e esperar que o *Windows* reconheça o *driver*. Para versões do MPLAB® seja inferior a 7.31 uma janela de instalação do dispositivo USB (placa ICD2 BR) irá aparecer. Se isso ocorrer localize o *driver* na pasta de instalação do MPLAB® (ICD2\Drivers ou Driversnn\ICD2_USB). Onde nn é a versão do sistema operacional.

Com a placa conectada à USB e devidamente instalada, abra o ambiente de desenvolvimento MPLAB® IDE. Vá em: **"*Configure >> Select Device*"** e selecione o PIC a ser utilizado, no nosso caso o PIC32MX360F512L. Perceba que na área ***"Programmers"*** o ICD2 aparece na cor verde, indicando que o mesmo pode ser utilizado para o PIC em questão. O mesmo é valido para **"*Debuggers*"**. Ou seja, a placa ICD2 BR pode ser utilizada tanto para programação quanto depuração de programas para o PIC32MX360F512L. (Figura 35)

O MPLAB® IDE fornece ainda um simulador (MPLAB® SIM) que é utilizado para testar o programa antes de gravá-lo no microcontrolador. Desta forma o usuário pode testar o programa mesmo na ausência da placa ICD2BR. Mais detalhes do MPLAB® SIM serão dados no Anexo IV : MPLAB® SIM. Perceba pela Figura 35 que a utilização deste componente é apenas parcial (bola amarela), ou seja, algumas das ferramentas do simulador podem não funcionar para o PIC32MX360F512.

[3] Atenção : Ao colocar o PIC32MX360F512L na placa EXPLORER16BR tome cuidado com a pinagem. Tanto na placa EXPLORER16 BR quando no PIC32MX360F512L existe uma pequena seta desenhada indicando o pino 1.

[4] O **Erro! Fonte de referência não encontrada.** mostra a forma de utilizar o *socket* em casos nos quais a placa de desenvolvimento não pode ser utilizada.

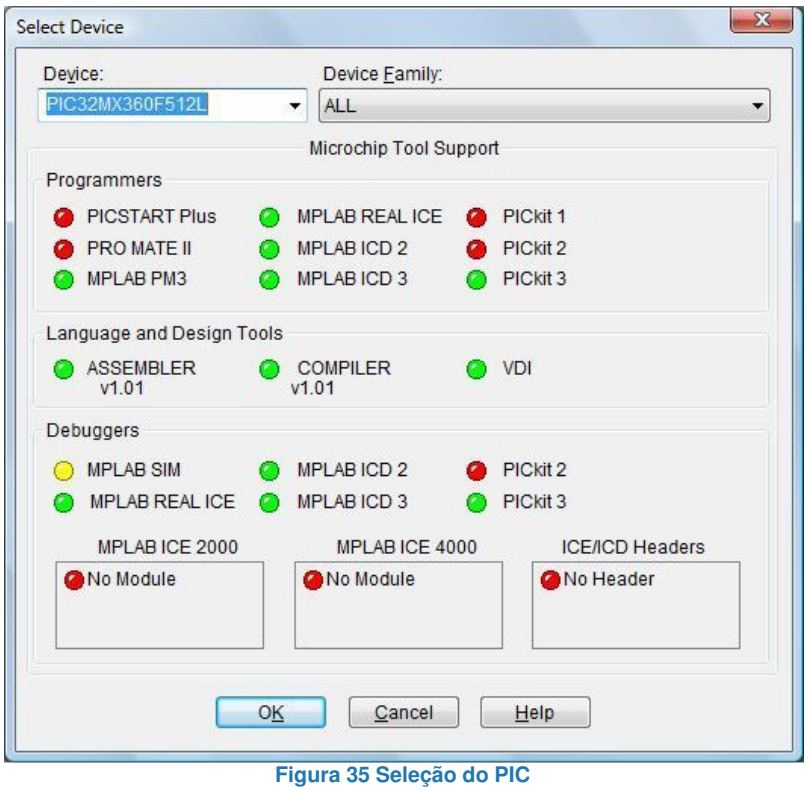

Figura 35 Seleção do PIC

Para utilizar a placa ICD2 BR vá em: ***"Programmer >> Select Programmer >> 2 MPLAB ICD2"***. Provavelmente uma mensagem indicando que a placa não está conectada irá aparecer. Para conectá-la clique em: ***"Reset and Connect to ICD"*** na barra de tarefas. (Figura 36)

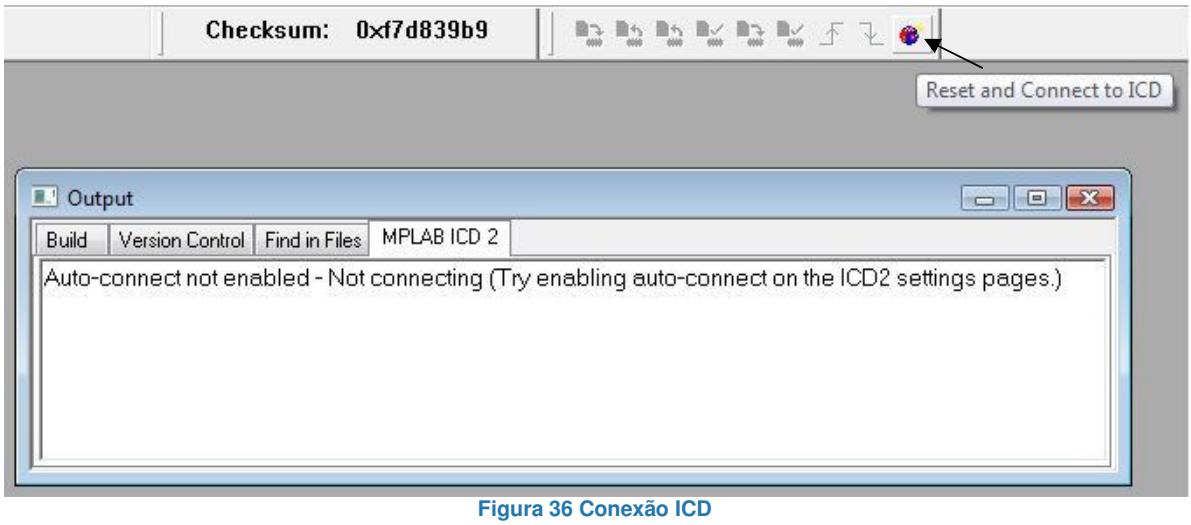

Figura 36 Conexão ICD

Uma mensagem poderá aparecer. Clique **ok.**

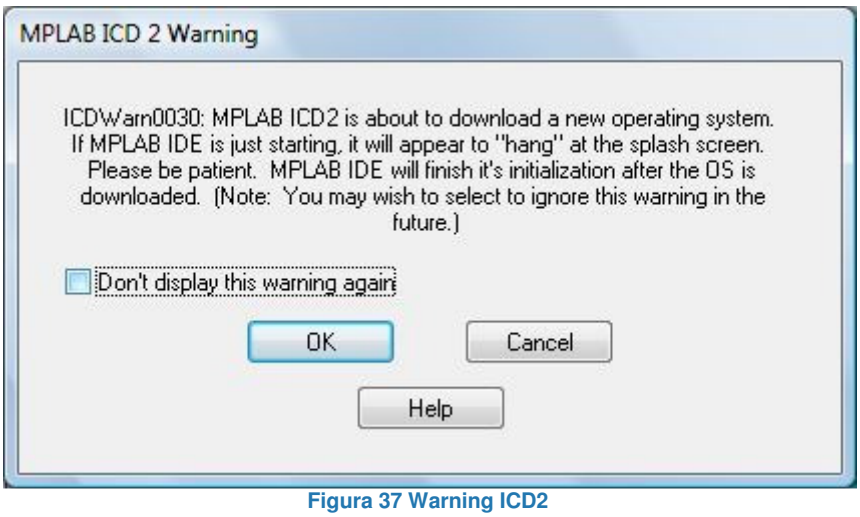

Figura 37 Warning ICD2

Pronto! A ICD2 BR foi instalada e está pronta para gravar/depurar o programa[5]. Para programar o seu código no PIC instalado na placa Explorer 16 BR, ou seja, descarregar o arquivo .hex com o código binário corresponde ao código em C que você compilou presione o ícone ***Program target device*** conforme indicado na Figura 38. Aguarde pelo processo de gravação e a mensagem final ***Program Succeeded***.

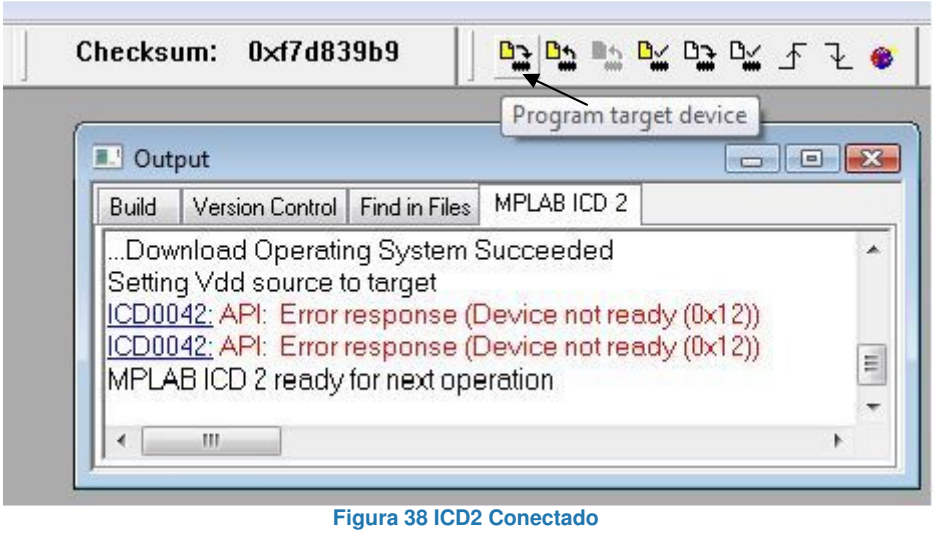

Figura 38 ICD2 Conectado

[5] Durante as aulas experimentais iremos mostrar como se grava um programa haja visto que até o momento não possuímos nenhum código para gravação.

7. Atividades Práticas

Como dito anteriormente no início da apostila, cada aula prática esta dividida em 3 atividades.

A atividade 1 é uma atividade simples com um código pronto para ser simulado e completamente funcional. Nessa atividade o aluno poderá atestar o funcionamento do periférico.

Na atividade 2 o aluno deverá ser capaz de realizar uma modificação na atividade 1. A terceira e última atividade exigirá que o aluno desenvolva um código por completo, baseado no que aprendeu na primeira e segunda atividades.

Cada uma das 7 aulas está organizada nas seguintes seções: Objetivos; Referências necessárias para execução da atividade; Introdução ao uso dos periféricos; Registradores usados na Programação; Atividade 1; Atividade 2 e Atividade 3.

7.1. Aula 1 – Configuração de Pinos de Entrada e Saída

Objetivo

Aprender a configurar a Interface que controla os pinos do MCU para Leitura de botões e acionamento de LEDs.

Referências

- *Datasheet* do MCU PIC32MX3XX-4XX. *Pin diagram* (pág. 4) e Capitulo 12.0: I/O Ports
- Manual do kit Explorer 16 BR. Seções 1.3 (Teclado) e 1.4 (LEDs)
- Esquema Elétrico dos Botões e LEDs da placa Explorer 16 BR.

Introdução

Um microcontrolador é um tipo de circuito integrado (*chip*). E todo *chip* possui pinos de entrada e saída. Alguns desses pinos tem funcionalidade específica e não estão disponíveis para que os usuários programe ou modifique o seu comportamento, isto é, um pino de saída não pode ser programado como entrada e vice-versa. Entretanto, a maior parte dos pinos de um microcontrolador pode ser configurado pelo usuário. Os pinos podem ser configurados como entrada ou saída, tipo digital ou analógico, definir pinos de saída com um valor lógico "1" ou com um valor lógico "0". Essa configuração dos pinos é realizada programando-se registradores específicos para tal fim.

O MCU PIC32MX360F512L que usaremos no nosso curso possuem pinos que podem ser programados pelo usuário e outros pinos com fuções bem definidas que não permitem programação. No *datasheet* do PIC32MX360F512L você encontrará todas essas informações.

No PIC32MX360F512L os pinos programáveis com uma mesma característica são agrupados em PORTOS (*I/O Ports*). Os PORTOS são portanto conjunto de pinos programáveis com uma mesma característica estrutural e elétrica. No *datasheet* do PIC32MX360F512L Capitulo 12.0: *I/O Ports* você encontrará mais informações.

Os PORTOS do PIC32MX360F512L recebem o nome de letras do alfabeto de A a G. Portanto, existe o PORTO A que é o conjunto de pinos RA0, RA1, RA2, RA3, até RA7. Vide *datasheet Pin diagram* (pág. 4). Assim como existe um PORTO B, um PORTO C, etc. Cada PORTO possuí uma quantidade de pinos própria e são atrelados a funcionalidades específicas do MCU. Os PORTOS são conhecidos também como GPIO (*General Purpose Input/Output*), ou seja, são portas (pinos do microcontrolador) para uso geral, como acionamento de algum circuito eletrônico ou leitura lógica (0 ou 1) de algum sensor, comando ou mesmo o pressionar de um botão.

Associado a cada PORTO, existe um conjunto de registradores. Cada registrador desse conjunto tem um nome que o distingue dos demais e uma funcionalidade específica. Exemplo: registrador TRIS associado aos pinos do PORTO A chama-se TRISA. Vamos agora descrever os registradores que usaremos para configurar os pinos programáveis desse MCU.

Registradores

Os principais registradores do microcontrolador que deveremos programar (configurar) para definir os pinos como entrada ou saída; pino digital ou analógico **nesta prática** são: LAT, TRIS e PORT.

Todos os pinos programáveis desse MCU possuem quatro registradores para a sua manipulação e configuração. Esses registradores são:

TRISx – Tem por objetivo guardar a configuração da direção do pino, isto é se é Entrada ou Saída. Colocar "1" em um bit **p** do registrador TRIS, significa que o pino **p** do PORTO x será uma entrada. Da mesma forma, configurar o mesmo bit **p** do registrador TRIS com "0" significa que queremos que tal pino seja uma saída.[6] Veja o exemplo de configuração do registrador TRISA a seguir. A configuração é feita em hexadecimal para facilitar a identificação se um pino esta sendo configurado como entrada ou saída.

Ex.: TRISA = 0xFF80; // 1111 1111 1000 0000 pinos RA0 até RA6 serão saída, os demais entradas.

PORTx – Esse registrador será usado para a leitura de um dado (valor lógico) do pino se o mesmo for configurado como um pino de **Entrada**.
Botões, chaves e entradas digitais são associadas a pinos de entrada. Os valores lógicos de botões, chaves e entradas digitais ficam registrados no registrador PORT na posição correspondente do pino do PORTO a qual estão ligados. Veja o exemplo a seguir.

Ex.: PORTB = 0xFFFE; // 1111 1111 1111 1110
Ler um nível lógico "1" de uma posição (bit) **q** do registrador PORTB, significa dizer que o pino **q** recebeu um nível lógico digital alto em sua entrada. Da mesma forma que se lermos "0", significará que o pino **q** está em nível lógico digital baixo.

LATx – Esse registrador será usado para guardar o que será escrito (valor lógico) em pino configurado como um pino de **Saída**.
LEDs e saídas digitais são associadas a pinos de saída. Os valores lógicos que se quer aplicar a LEDs e saídas digitais ficam registrados no registrador LAT na posição correspondente do pino do PORTO a qual estão ligados. Veja o exemplo a seguir.

Ex.: LATC = 0xFFFE; // 1111 1111 1111 1110
Escrever um nível lógico "1" de uma posição (bit) **q** do registrador PORTB,, significa que esse pino **q** receberá um nível lógico digital alto em sua saída. Escrever "0" significará nível lógico digital baixo.

ODCx – Esse registrador será usado para configurar um pino saída como dreno aberto ou não.
Nessa prática não usaremos esse registrador. O aluno que desejar maiores informações deverá consultar o *datasheet* do MCU.

[6] x' representa a letra do PORTO em questão. Exemplo: TRISA, TRISB, TRISC, …, TRISG.
O número de PORTOS varia de um microcontrolador PIC para outro.

Atividade 1

Nesta atividade iremos configurar pinos de Entrada e Saída (E/S) de um MCU para utilização dos botões e leds. Apresentaremos um pequeno programa para acionamento de um led através de um botão da placa de desenvolvimento Explorer 16 BR.

Crie um **novo projeto** no ambiente MPLAB.
Crie um arquivo **"main.c"**.

Adicione o(s) cabeçalho(s) (***includes***) mínimos necessários a todo o projeto.

```
// INCLUDES
#include <p32xxxx.h> //include para o PIC32MX360F512L
```

Em seguida, no mesmo código em C, configure o modo de gravação.[7]

```
// CONFIGURACAO PARA GRAVACAO
#pragma config FPLLMUL = MUL_16, FPLLIDIV = DIV_2, FPLLODIV = DIV_8, FWDTEN = OFF
#pragma config POSCMOD = HS, FNOSC = PRI, FPBDIV = DIV_2
```

Cada registrador do PIC possui um endereço de memória específico e um espaço de 32bits. Para o PIC32MX360F512L esses endereços estão mapeados no arquivo *header* ***"MPLAB® C32 >> pic32-libs >> include >> proc >> p32mx360f512l.h"***.

Cada linha da cláusula ***"#define"*** no código abaixo define um nome para cada endereço de registrador do microcontrolador que pode ser configurado pelo programador. Para melhor entendimento do código, atribuímos nomes representativos às entradas (botões) e saídas (leds) e atribuímos respectivamente a pinos correspondentes de acordo com o Esquema Elétrico do kit Explorer 16 BR (vide referências dadas).

Dessa forma temos: #define nome endereço

Assim, ao invés de ***"PORTDbits.RD6"*** podemos utilizar, por exemplo, ***"Botao_1"*** para facilitar a compreensão do código e o torná-lo mais legível. O mesmo é válido para todos os outros ***defines*** listados a seguir.

```
// 0 Pressionado / 1 Liberado
#define Botao_1 PORTDbits.RD6 //BOTÃO 1
#define Botao_2 PORTDbits.RD7 //BOTÃO 2
#define Botao_3 PORTAbits.RA7 //BOTÃO 3
#define Botao_4 PORTDbits.RD13 //BOTÃO 4

// 0 Apagado / 1 Aceso
#define Led1 LATAbits.LATA0 //LED1
#define Led2 LATAbits.LATA1 //LED2
#define Led3 LATAbits.LATA2 //LED3
#define Led4 LATAbits.LATA3 //LED4
#define Led5 LATAbits.LATA4 //LED5
#define Led6 LATAbits.LATA5 //LED6
#define Led7 LATAbits.LATA6 //LED7
```

[7] Esta configuração indica a freqüência de operação do seu microcontrolador. Por ora não se preocupe com o entendimento do código nas duas linhas abaixo. Mais detalhes serão dados em uma atividade futura.

Crie a função int **main(void)**.

Limpe (dê um *reset*) todas os pinos dos PORTOS. Isso é necessário, pois outro programa pode estar gravado na memória do MCU e isto evita possíveis comportamentos não previstos.

```
// Reset
LATA = 0;
LATB = 0;
LATC = 0;
LATD = 0;
LATF = 0;
LATG = 0;
```

Conforme dito anteriormente o registrador TRIS é responsável por configurar se um pino irá operar como entrada (1) ou saída (0). Cada um dos 7 registradores TRIS (TRISA, TRISB... TRISG) possui 32 bits porém, somente os 16 bits menos significativos (bits de 0 a 15) estão disponíveis para configuração do usuário conforme pode ser observado no capítulo 12 do *datasheet* do PIC.

Usualmente os registradores são configurados através de números hexadecimais por razões de clareza do código escrito. No código abaixo o comentário mostra o valor dos 16 bits menos significativos, equivalente a escrita hexadecimal do respectivo registrador, a título de melhor visualização de quais pinos são entradas (1) e quais são saídas (0).

Para configurar a direção dos pinos de E/S insira o código abaixo.[8]

```
// Configuração da direção dos pinos de I/O's. (0-Output 1-Input)
DDPCONbits.JTAGEN = 0;

TRISA = 0xFF80; // 1111 1111 1000 0000 Leds: PORT A0-A6 (Output), Botao_3: PORT A7 (Input)
TRISB = 0xFFFF;
TRISC = 0xFFFF;
TRISD = 0xFFFF; // 1111 1111 1111 1111 Botoes: PORT D6,D7,D13 (Input)
TRISE = 0xFF00;
TRISF = 0xFFFF;
TRISG = 0xFEBF;
```

Pronto. A configuração dos botões e LEDs esta feita.

Para testar crie um *loop* infinito e faça com que o Botao_1 acenda o LED1, o Botao_2 acenda o LED2, o Botao_3 o LED3 e o Botao_4 o LED4.

```
while(1)
{
   if(!Botao_1) Led1 = 1; //TESTE BOTÃO 1
   else Led1 = 0;

   if(!Botao_2) Led2 = 1; //TESTE BOTÃO 2
   else Led2 = 0;

   if(!Botao_3) Led3 = 1; //TESTE BOTÃO 3
   else Led3 = 0;

   if(!Botao_4) Led4 = 1; //TESTE BOTÃO 4
   else Led4 = 0;
}
```

[8] No final do material existe um capítulo denominado erratas cujo capítulo 1 (Errata I : Esquemático Leds e Botoes) demonstra o porquê de utilizar a porta RA7 como Botão_3 e não como LED8.

Compile o programa (**Ctrl+F10**). Se você executou tudo corretamente uma mensagem dizendo que o programa foi corretamente compilado irá aparecer. (**Erro! Fonte de referência não encontrada.**Figura 39)

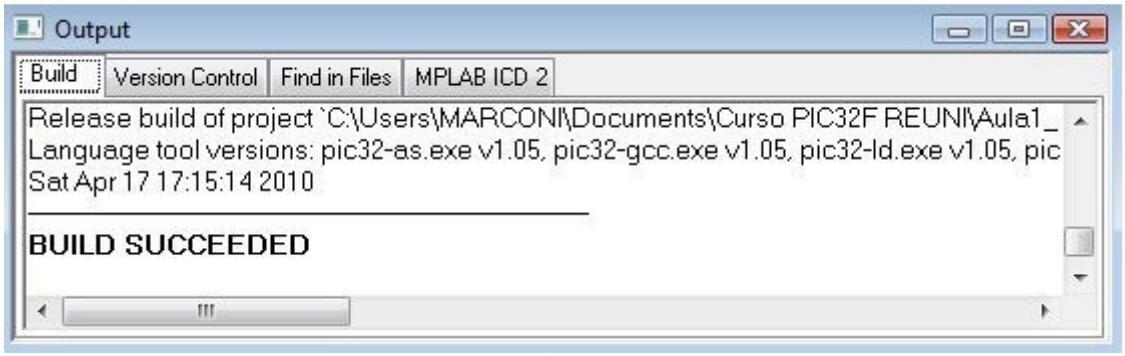

Figura 39 Compilação correta do programa

Antes de gravar o programa no PIC que está no kit Explorer 16 BR é interessante simulá-lo (testá-lo) a fim de verificar se o programa está funcionando como o esperado. A simulação e validação do seu código deverá ser feita usando como ferramenta o simulador MPLAB® SIM cujo modo de operação é mostrado no Anexo IV : MPLAB® SIM.

O complemento dessa atividade, ou seja, gravação com o ICD2 e teste na placa EXPLORER 16 BR deverão ser realizados em laboratório sob o acompanhamento do professor. Após verificar o correto funcionamento do programa com o MPLAB® SIM conecte o kit EXPLORER 16 BR na placa ICD2 BR através do cabo RJ12.

Conecte a placa ICD2 BR ao computador através da porta USB.

Alimente o kit EXPLORER16 BR com a fonte de alimentação.

Para gravar o programa no PIC com o ICD2 BR vá em: ***"Programmer >> Select Programmer >> MPLAB® ICD 2"***.

Na barra de tarefas clique em: ***"Reset and Connect to ICD"***. Se a placa estiver corretamente instalada a seguinte mensagem irá aparecer: (Figura 40)

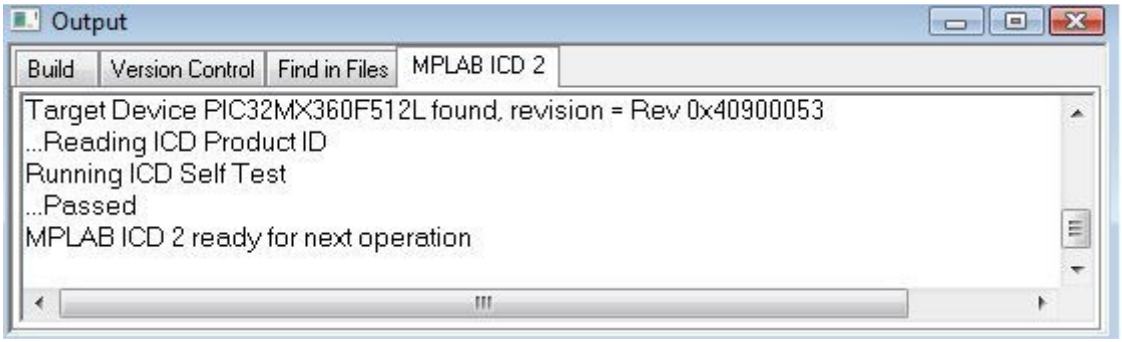

Figura 40 ICD2BR Corretamente conectado

Para gravar o programa clique em: **"*Program Target Device*"**.

Se a gravação foi bem sucedida a seguinte mensagem irá aparecer: (Figura 41)

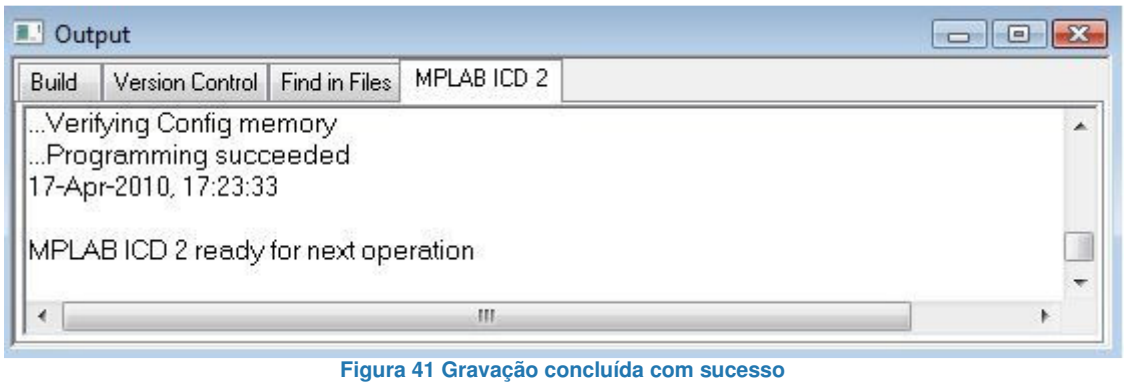

Figura 41 Gravação concluída com sucesso

Para testar o programa desconecte a placa ICD2 BR do kit EXPLORER 16 BR, pressione os botões e veja o que acontece.[9]

Atividade 2

Modifique o programa 1 para que o Botao_1 acenda o LED 1 e este se apague somente se o botão 2 for pressionado.

Atividade 3

Quando o Botao_1 for pressionado os LEDS deverão acender na seguinte seqüência (LED7, LED6, LED5, LED4, LED3 LED2, LED1, LED7, LED6...), ou seja, da direita para esquerda.
Quando o Botao_2 for pressionado os LEDS deverão acender da esquerda para direita.
Quando o Botao_3 for pressionado todos os LEDS deverão piscar.
Quando o Botao_4 for pressionado todos os LEDS deverão se apagar.

Dica: Utilize um *loop for* de 0 a 30000 como mostrado no trecho de código abaixo, para gerar um atraso suficiente, de forma que o usuário possa conseguir ver os LEDs piscarem na ordem proposta. Observe o código a seguir.

```
int main(){
   while(1){
      // AQUI VEM O PROGRAMA
      for (tempo=0; tempo<20000; tempo++); // temporização
   } // Fim While
} // Fim Main
```

[9] No final do material existe um capitulo denominado erratas cujo capitulo 2 (Errata II : Efeito de carga do ICD2BR) demonstra o porquê de desconectar a placa ICD2 BR do kit EXPLORER16BR para testar o programa.

7.2. Aula 2 – Configuração do Periférico de Interrupções de Eventos Externos por Notificação de Mudança de Estado.

Objetivo

Aprender o conceito de interrupção e configurar a interface de tratamento de interrupções externas do MCU PIC32MX360F512L.

Referências

– *Datasheet* do MCU PIC32MX3XX-4XX. *Pin diagram* (pág. 4) e Seção 12.2.9.

Introdução

Uma interrupção pode ser definida como um sinal assíncrono externo a CPU, indicando a necessidade de atenção (ou a realização de alguma ação) da mesma à ocorrência desse evento. Os pedidos de interrupção à CPU de um MCU podem ser realizados por algum periférico do próprio MCU ou por um sinal elétrico externo gerado por outro sistema digital ligado a MCU através de qualquer um de seus pinos de entrada. Nessa aula nos dedicaremos à configuração de Interrupções de Sinais Elétricos Externos ao MCU sensíveis através a mudança de estado dos pinos de E/S do mesmo.

Um ciclo de instrução de uma CPU com consulta a requisições de interrupções consiste nos seguintes passos sequenciais:

1. Busca da Instrução.
2. Decodificação e incremento do endereço de busca da próxima instrução.
3. Busca dos operandos demandados pela instrução.
4. Execução da operação demandada pela instrução.
5. Escrita dos resultados da operação.
6. Verificação se houve pedido de interrupção, salvar o endereço de retorno na pilha e atualização do endereço de busca da nova instrução com o endereço da rotina de tratamento da interrupção.

Evidentemente que, para que o passo 6 ocorra, tem de existir no *hardware* um registrador dedicado para armazenar os pedidos de interrupção realizados pelos periféricos e sinais elétricos externos. Esse registrador será consultado pela CPU durante a execução desse passo. Caso haja algum pedido de interrupção, a CPU salvará o endereço armazenado no registrador PC (*Program Counter*), ou seja o endereço de retorno, na pilha e atualizará o PC com o endereço da rotina de tratamento de interrupção do periférico que pediu a atenção. E o ciclo de instruções continuará a partir da primeira instrução da rotina (função) de tratamento da interrupção do periférico que pediu a atenção da CPU.

Em muitas aplicações de sistemas embarcados necessita-se de uma interrupção externa para informar a CPU do microcontrolador que algum periférico acabou de fazer sua tarefa e está pedindo a atenção dela ou que algum evento externo através de um pino de entrada ocorreu.

O periférico ou interface responsável pelo uso das interrupções externas está diretamente relacionado a certos pinos do MCU e pode ser configurado para gerar interrupções por mudança de estado (Ex.: de um nível lógico de 0 para 1 ou o contrário) dentre várias outras formas que são descritas no *datasheet* do MCU. Esse recurso é muito usado quando queremos ler se um botão foi pressionado, ler teclados, dar atenção a uma chegada de dados de um dispositivo externo ou qualquer outro evento assíncrono a qualquer momento que eles ocorrerem e ainda sincronizar eventos com a rede elétrica, entre outros exemplos.

Na aula 1 um evento ocorrido em uma porta digital (ex.: pressionar de um botão) só era tratado quando o programa chegasse na linha que fazia a leitura do pino digital. Com a configuração

do periférico ou interface que dá atenção as interrupções externas, a ocorrência de alguma mudança de nível em um pino digital provocará um pedido de interrupção a CPU e gerará um salto incondicional para a rotina de tratamento de interrupção específica.

Registradores

Para configurar a interface do MCU que dá atenção ao pedido de interrupção a um evento ocorrido em um dos pinos externos citados precisaremos programar os seguintes registradores:

CNCON (*Change Notification Control Register*): Registrador de CONtrole ou CONfiguração de interrupções que habilita ou desabilita o periférico de interrupção por mudança de estado e sua habilidade de produzir interrupção. O bit ON (décimo quinto bit desse registrador) é o bit responsável por habilitar ou não esse tipo de interrupção. ON = 1 (habilitado), ON = 0 (contrário).

CNEN (*Change Notification Enable Register*): Registrador responsável por habilitar ou desabiltar os pinos que responderão pela interrupção por mudança de estado no MCU.

CNPUE (*Change Notification Pull-Up Enable Register*): Registrador responsável por habilitar ou desabiltar um resistor de *pull-up* para os pinos que responderão pela interrupção por mudança de estado no MCU. O ato de habilitar um pino com resistor de *pull-up* interno funciona como uma fonte de corrente que está ligado ao pino, e elimina a necessidade de resistores externos para botões ou dispositivos de teclado que estejam conectados a tais pinos. Quando uma porta for configurada com saída digital esse recurso tem que ficar desabilitado.

IEC1 (*Interrupt Enable Control Register 1*): Registrador que armazena o estado de disponibilidade de atenção da CPU a ocorrência de uma interrupção por mudança de estado. Exemplo: CNIE = 1; CNIE é um bit do registrador IEC1 que programado com nível lógico 1 significa que a CPU atenderá interrupções de mudança de estado quando estas acontecerem.

IFS1 (*Interrupt Flag Status Register 1*): Registrador que armazena o estado pedido de atenção de um evento por mudança de estado à CPU. Exemplo: CNIF é um bit do registrador IFS1 que se estiver em nível lógico 1 significa que um evento externo causado por mudança de nível em algum pino de Entrada/Saída programado para isso aconteceu e pede a atenção da CPU para realizar alguma ação.

IPC6 (*Interrupt Priority Control Register 6*): Registrador que armazena o nível de prioridade do tratamento do pedido de atenção do evento externo por mudança de estado à CPU. Exemplo: Se o conjunto de bits CNIP = 1 for programado por você no do registrador IPC6, significará que o periférico (no caso o sinal externo) será atendido segundo o nível de prioridade 1 (em uma escala de 0 a 7, sendo 7 a de maior prioridade) pela CPU quando houver um evento externo.

Para configurar o periférico de interrupção por mudança de estado associado a pinos de entrada/saída do MCU PIC32MX360F512L deveremos programar os seguintes passos:

1) Desabilitar pedidos de interrupção à CPU.
2) Configurar a entrada de notificação de mudança de estado correspondente a entrada no registrador TRISx. **Observação**: Se o pino de E/S puder ser configurado com uma entrada analógica será necessário programar também o bit correspondente AD1PCFG = 1 para garantir que o Pino de E/S seja uma entrada digital.
3) Habilitar o periférico de notiifcação de mudança de estado no registrador CNCON (bit ON).
4) Habilitar as entradas que serão utilizadas no registrador CNEN.

5) Se necessário, habilitar os resistores de *pull-up* das entradas configuradas no passo anterior configurando através do registrador CNPUE.
6) Limpar a *flag* de interrupção por mudança de estado, ou seja, o bit CNIF do registrador IFS1.
7) Configurar a prioridade de interrupção por mudança de estado no registrador IPC6 (bits CNIP).
8) Habilitar a interrupção por mudança de estado no registrador IEC1 (bit CNIE).
9) Habilitar pedidos de interrupção à CPU.
10) Na rotina de tratamento de interrupção, para saber qual pino gerou a interrupção, basta fazer uma leitura no PORTO correspondente a entrada selecionada.

Atividade 1.a

Em **"main.c"** insira o código para a atividade 1a.

```
1    // INCLUDES
2    #include <p32xxxx.h> //include para o PIC32MX360F512
3    #include <plib.h>
4
5    // CONFIGURACAO PARA GRAVACAO
6    #pragma config FPLLMUL = MUL_16, FPLLIDIV = DIV_2, FPLLODIV = DIV_8, FWDTEN = OFF
7    #pragma config POSCMOD = HS, FNOSC = PRI, FPBDIV = DIV_8 //(PBCLK is SYSCLK divided by 8,4,2,1)
8
9    // DEFINES
10   // BOTOES (0:PRESSIONADO 1:LIBERADO)
11   #define Botao_1 PORTDbits.RD6 //BOTÃO 1
12   #define Botao_2 PORTDbits.RD7 //BOTÃO 2
13   #define Botao_3 PORTAbits.RA7 //BOTÃO 3
14   #define Botao_4 PORTDbits.RD13 //BOTÃO 4
15   // LED (0:APAGADO 1:ACESO)
16   #define Led1 LATAbits.LATA0 //LED1
17   #define Led2 LATAbits.LATA1 //LED2
18   #define Led3 LATAbits.LATA2 //LED3
19   #define Led4 LATAbits.LATA3 //LED4
20   #define Led5 LATAbits.LATA4 //LED5
21   #define Led6 LATAbits.LATA5 //LED6
22   #define Led7 LATAbits.LATA6 //LED7
23   //#define Led8 LATAbits.LATA7 //LED8 // Utilizar como Botao_3 e nao Led_8
24
25   int main(){
26       int i=0;
27
28       // Reset
29       LATA = 0;
30       LATB = 0;
31       LATC = 0;
32       LATD = 0;
33       LATF = 0;
34       LATG = 0;
35
36       DDPCONbits.JTAGEN = 0;
37
38       // Configuração da direção dos pinos de I/O's. (0-Output 1-Input)
39
40       TRISA = 0xFF80; // 1111111110000000 Leds: PORT A0-A6 (Output), Botao_3: PORT A7 (Input)
41       TRISB = 0xFFFF;
42       TRISC = 0xFFFF;
43       TRISD = 0xEFCF; // 1110111111001111 Botoes: PORT D6,D7,D13 (Input)
44       TRISE = 0xFF00;
45       TRISF = 0xFFFF;
46       TRISG = 0xFEBF;
47
48       while(1)
49       {
50
51           for (i=0; i<50000; i++);
52           for (i=0; i<50000; i++);
53           for (i=0; i<50000; i++);
54           for (i=0; i<50000; i++);
55
56           if(!Botao_3)
57               Led3=!Led3;
58       }
59
60   }
```

Este programa foi inserido para percebemos melhor a utilização da interrupção. Note que a leitura do botão 1 só será feita após a execução dos 4 loops *for*, utilizados somente para provocar um atraso de tempo significativo na execução do programa de forma que o usuário perceba que o acender do LED não ocorre de forma imediata.

1. Teste o programa com o MPLAB SIM®.
2. Após testar o programa conecte o kit EXPLORER 16 BR na placa ICD2 BR através do cabo RJ12.

3. Conecte a placa ICD2 BR ao computador através da porta USB.
4. Alimente o kit EXPLORER 16 BR com a fonte de alimentação.
5. Compile o programa (**Ctrl+F10**).
6. Grave o programa no MCU do kit EXPLORER 16 BR (**Program Target Device**).
7. Teste o programa.

Perceba que o LED 3 não acende no momento em que o botão 3 é pressionado.

Atividade 1.b

Em **"main.c"** insira o código para a atividade 1b.

```
1   // INCLUDES
2   #include <p32xxxx.h>                    //include para o PIC32MX360F512
3   #include <plib.h>
4
5   // CONFIGURACAO PARA GRAVACAO
6   #pragma config FPLLMUL = MUL_16, FPLLIDIV = DIV_2, FPLLODIV = DIV_8, FWDTEN = OFF
7   #pragma config POSCMOD = HS, FNOSC = PRI, FPBDIV = DIV_8 //(PBCLK is SYSCLK divided by 8,4,2,1)
8
9   // DEFINES
10  // Ver Errata : Esquematico Leds e Botoes
11
12  // BOTOES (0:PRESSIONADO 1:LIBERADO)
13  #define        Botao_1  PORTDbits.RD6          //BOTÃO 1
14  #define        Botao_2  PORTDbits.RD7          //BOTÃO 2
15  #define        Botao_3  PORTAbits.RA7          //BOTÃO 3
16  #define        Botao_4  PORTDbits.RD13         //BOTÃO 4
17
18  // LED (0:APAGADO 1:ACESO)
19  #define Led1   LATAbits.LATA0          //LED1
20  #define Led2   LATAbits.LATA1          //LED2
21  #define Led3   LATAbits.LATA2          //LED3
22  #define Led4   LATAbits.LATA3          //LED4
23  #define Led5   LATAbits.LATA4          //LED5
24  #define Led6   LATAbits.LATA5          //LED6
25  #define Led7   LATAbits.LATA6          //LED7
26  //#define Led8  LATAbits.LATA7          //LED8  // Utilizar como Botao_3 e nao Led_8
27
28  void __ISR(_CHANGE_NOTICE_VECTOR, ipl5) CN_Interrupt_ISR(void)
29  {
30  unsigned int value;
31  value = PORTD;   // ver datasheet pg 327
32
33  Led1=!Led1;
34
35  IFS1bits.CNIF = 0; // clear the interrupt flag
36  }
37
38  int main(){
39          int i;
40          unsigned int value;
41
42          // Reset
43          LATA = 0;
44          LATB = 0;
45          LATC = 0;
46          LATD = 0;
47          LATF = 0;
48          LATG = 0;
49
50          // Configuração da direção dos pinos de I/O's. (0-Output 1-Input)
51          DDPCONbits.JTAGEN = 0;
52
53          TRISA = 0xFF80; // 1111111110000000 Leds: PORT A0-A6 (Output), Botao_3: PORT A7 (Input)
54          TRISB = 0xFFFF;
55          TRISC = 0xFFFF;
56          TRISD = 0xEFCF; // 1110111111001111 Botoes: PORT D6,D7,D13 (Input)
57          TRISE = 0xFF00;
58          TRISF = 0xFFFF;
59          TRISG = 0xFEBF;
60
61          value = PORTD; // ver datasheet pg 327
```

```
62
63          // CHANGE NOTIFICATION (CN)
64          CNCON = 0x8000; // Enable Change Notification Module, Continue Operation in Idle Mode
65          CNEN = 0x8000; //CN15 (RD6) enabled
66          CNPUE = 0x8000; //CN15 (RD6) pull-up Enabled
67
68          // Change Notification Interrupt
69          IEC1bits.CNIE =0; /// CN Interrupt Disabled
70          IFS1bits.CNIF = 0; //// CN Interrupt Request Flag
71
72          IPC6bits.CNIP = 7; //Priority 7 (0,1,...7)
73          IPC6bits.CNIS = 0; //SubPriority 0 (0,1,2,3)
74
75          IEC1bits.CNIE =1; /// CN Interrupt Enabled
76          INTEnableSystemMultiVectoredInt();
77
78
79          while(1)
80          {
81
82             for (i=0; i<50000; i++);
83             for (i=0; i<50000; i++);
84             for (i=0; i<50000; i++);
85             for (i=0; i<50000; i++);
86             if(!Botao_3)
87                     Led3=!Led3;
88
89          }
90
91   }
```

O programa da **atividade 1a** foi mantido para facilitar a comparação do efeito do atendimento a interrupção externa. Perceba, na linha 65 do código acima, que a interrupção que iremos utilizar é a CN15. Esse tipo de interrupção está associada a ISR (*Interrupt Service Routine*) chamada pela função: void __ISR(_CHANGE_NOTICE_VECTOR, ipl5) CN_Interrupt_ISR(void). Quando uma borda de subida ou de descida (transição de nível lógico 0 para 1 ou nível lógico 1 para 0, respectivamente) é percebida no pino RD6 (CN15) uma interrupção é gerada e a função CN_Interrupt_ISR é chamada. Nessa função deve ser inserida toda a sequência de ações a ser executada quando a interrupção externa for gerada. No nosso caso temos somente uma ação, mostrada na linha 33, que troca o valor do LED 1 a cada chamada da interrupção.

O código das linhas 31 e 61 (value = PORTD;) é um artifício para poder resetar o flag de pedido de interrupção do módulo change notification. Em outras palavras, sempre antes de limpar o flag a porta correspondente deve ser resetada para evitar erros de leitura do MCU durante as transições. Para maiores informações veja o texto explicativo na página 327 do *datasheet*.

1. Teste o programa com o MPLAB SIM®.
2. Após testar o programa conecte o kit EXPLORER 16 BR na placa ICD2 BR através do cabo RJ12.
3. Conecte a placa ICD2 BR ao computador através da porta USB.
4. Alimente o kit EXPLORER 16 BR com a fonte de alimentação.
5. Compile o programa (**Ctrl+F10**).
6. Grave o programa (***Program Target Device***).
7. Teste o programa.

Perceba neste caso que o LED 1 altera seu valor imediatamente após as transições de nível de sinal lógico de 0 para 1 e de 1 para 0 aplicada ao pino RD6 (representado pelo pressionar do botão no kit), enquanto que o LED 3 ainda deve esperar o programa executar os quatro *loops* ***for*** para acender.

Atividade 2

Modifique o programa da **Atividade** 1 b para que a ocorrência de uma transição de 0 para 1 faça o LED 1 da placa acender e o LED 2 apagar e que uma transição de 1 para 0 faça o LED 2 acender e o LED 1 apagar.

Atividade 3

Habilite dois canais de interrupção (CN15 e CN16). Crie, dentro da rotina de interrupção, um algoritmo capaz de distinguir qual dos dois canais foi o responsável pela chamada de interrupção.

Seu programa deverá executar uma tarefa (acender LED) diferente para cada um dos canais (CN15 ou CN16) somente na transição de nível lógico 0 para nível lógico 1.

7.3. Aula 3 - Configuração de Periféricos Contadores (Temporizadores ou *Timers* e Contadores de eventos externos ou *Counters*).

Objetivo

Aprender a utilizar e configurar *timers* e contadores de um MCU para sincronização de operações.

Referências

- *Datasheet* do MCU PIC32MX3XX-4XX. Capítulos 13.0 *Timer1* e 14.0 *Timers 2,3,4,5*.
- *Datasheet* – PIC32MX *Family Reference Manual* - consulte a seção 14.0.
- Aula 1 e Aula 2

Introdução

O microcontrolador PIC32MX360F512L possui 5 periféricos contadores de 16 bits, que podem ser configurados independentemente para trabalharem como temporizadores (*timers*, ou contadores de tempo), como contadores (contadores de eventos externos, i.e.: número de vezes que um botão for pressionado), ou ainda como acumuladores (medidores de larguras de pulsos) dependendo da aplicação que se quer desenvolver. Nessa aula configuraremos o periférico somente como contador de tempo (modo de operação como temporizador ou *timer*).

Os cinco periféricos contadores do PIC32MX360F512L são agrupados em duas categorias: periféricos contadores da categoria A e periféricos contadores da categoria B. Cada categoria A e B possuem características próprias de funcionamento (consulte a seção 14.0 do *datasheet – Family Reference Manual* para observar mais informações sobre as características de cada categoria). Alguns dos contadores pertencem a categoria A os demais pertencem a categoria B. Cada categoria de contadores possuem características próprias, tais como, modo de operação (contador, temporizador ou acumulador), limite de contagem, funcionamento com clocks de diferentes frequencias, operação em modo assíncrono com uma fonte de clock externa, funcionamento com ou sem interrupção, entre outras. Todas essas funcionalidades devem ser consultadas e estudadas no *datasheet* do MCU, mais especificamente nos capítulos 13.0 e 14.0.

Para contagem de tempo consideraremos que o contador será configurado no modo de contagem de tempo (temporizador ou *timer*). Quando o periférico estiver configurado para contar tempo adotaremos o termo **temporizador** ou (***Timer***). Para contagem de eventos adotaremos o termo **contador** (***Counter***). Para medir larguras de pulsos adotaremos o termo **acumulador** (***Accumulator***).

Ainda como caraterística desse MCU os contadores 2 e 3 ou 4 e 5 podem ser agrupados de dois em dois para formar um contador maior de 32 bits.

Registradores

Para configurar esse periférico do microcontrolador PIC32MX360F512L necessitaremos programar os seguintes registradores:

TxCON: Registrador de CONtrole ou CONfiguração associado ao contador x.

TMRx: Registrador que armazena a contagem. É um registrador de 16 bits. Ele é associado ao contador x;

PRx: Registrador de Período associado ao contador x;

Vamos agora detalhar cada um dos três registradores citados acima.

O periférico contador pode operar de três modos diferentes, como dito anteriormente, modo de operação *Timer*, modo de operação *Counter*, modo de operação *Accumulator*. Para escolher o modo de operação devemos configurar os bits TCS e TGATE do registrador TxCON conforme a tabela I retirada do *datasheet* do MCU.

Tabela I – Modo de Operação do Periférico Contador

Modo de operação	TxCON<TCS>	TxCON<TGATE>
Timer	0	0
Counter	1	X
Accumulator	0	1

Para ligar ou desligar o periférico contador tenho que programar o bit TON do registrador TxCON. Se faço TON = 1 ligo o periférico para contar, se faço TON = 0 o desligo.

O modo de operação de contagem de tempo (ou *Timer*) funciona obedecendo a Equação I:

$$Tempo\ de\ contagem\ desejado\ (segundos) = \frac{PRx}{Fosc} \times FPDIV \times PS \qquad \text{(Equação I)}$$

Na Equação I, PRx representa o valor que deverá ser programado no registrador de período para que o tempo de contagem desejado (em segundos) seja alcançado. Fosc é a frequencia do oscilador externo (da placa Explorer 16 BR) usado como sinal de clock para funcionamento do MCU. FPDIV é um parâmetro de configuração do MCU que tem efeito **somente na gravação** do programa no MCU. O parâmetro FPDIV serve para dividir a frequencia do clock externo (Fosc) por um valor inteiro, no intuito de gerar um clock de frequencia menor para funcionamento do contagem. Chameremos esse clock proveniente da divisão por FPDIV de clock interno. PS é um recurso do periférico contador chamado de *Prescaler*. O *Prescaler* nada mais é que um fator de multiplicação de contagem de ciclos de clock. Suponha que o prescaler esteja configurado como fator multiplicativo 1:4. Isso significa que o periférico contador **x** vai contar de um em um (através do registrador TMRx) a cada 4 ciclos de clock interno do MCU. O *Prescaler* ou PS é configurado atribuindo-se um valor inteiro ao conjunto de bits TCKPS do registrador TxCON. Cada valor inteiro corresponde a um fator multiplicativo de contagem. O valor do *Prescaler* varia de contador para contador e sua configuração tem que ser consultada no *datasheet*.

Para cada contador teremos também bits relativos ao controle de interrupção, pois o contador é um periférico que trabalha independentemente da CPU. Uma interrupção de *timer* (contador de tempo) ou do contador de eventos sempre ocorrerá na CPU do microcontrolador, toda vez que uma contagem chegar ao fim (o fim de contagem dependerá da forma como os parâmetros da Equação I estiverem configurados).[10] Os registradores envolvidos no controle de interrupções do periférico contador são:

IEC0 (*Interrupt Enable Control Register 0*): Registrador que armazena o estado da atenção da CPU para a ocorrência de uma interrupção em qualquer contador **x**. Exemplo: T1IE = 1; T1IE é um bit do registrador IEC0 que programado com nível lógico 1 significa que a CPU atenderá interrupções do *Timer* 1 (T1) quando esse tipo de interrupção acontecer.

IFS0 (*Interrupt Flag Status Register 0*): Registrador que armazena o estado pedido de atenção de um periférico contador **x** à CPU. Exemplo: Suponha que o bit T1IF = 1 do registrador IFS0; T1IF é um bit do registrador IFS0 que se estiver em nível lógico 1 significa que o periférico atingiu o fim da contagem para qual foi programado e pede a atenção da CPU para o evento de fim de contagem. Portanto IFS0 é um registrador que a CPU deve consultar ao final de cada ciclo de instrução para verificar se algum periférico lhe pediu a atenção.

IPCx (*Interrupt Priority Control Register x*): Registrador que armazena o nível de prioridade do tratamento do pedido de atenção de um periférico contador **x** à CPU. Exemplo: Se o conjunto de bits T1IP = 1 for programado por você no do registrador IPCx, significará que o periférico será atendido segundo o nível de prioridade 1 (em uma escala de 0 a 7, sendo 7 a de maior prioridade) pela CPU quando houver um evento de fim de contagem.

[10] O registrador pode ser configurado com 16 bits ou utilizar a combinação de dois contadores (*timers*) de 16 bits cada um para formar um contador (*timer*) de 32 bits.

Atividade 1.a

Nessa atividade programaremos o periférico contador 5 do PIC32MX360F512L no modo de operação de um *Timer*. De início, vamos escrever uma função que chamaremos de **delay_1ms(int x)** que tem por objetivo produzir um atraso preciso de 1 ms para o desenvolvimento de qualquer aplicação com esse MCU que exija precisão de tempo da ordem de milisegundos. Essa função não retorna nenhum valor (void), só provoca a ocorrência de uma interrupção na CPU quando a contagem programada é atingida. Essa função demanda a passagem de um parâmetro inteiro **x** que representa o número de milisegundos que desejaremos que o periférico *timer* conte até gerar a interrupção.

Nesta atividade iremos fazer o LED1 acender a durante 2 segundos toda vez que o Botao_1 for pressionado utilizando o contador do *Timer5*.

Crie um **novo projeto**.
Crie um arquivo **"main.c"**.

Adicione os includes.

```
// INCLUDES
#include <p32xxxx.h> //include para o PIC32MX360F512L
#include <pblib.h>
```

Configure o modo de gravação. O trecho da linha destacado no comentário do código abaixo, indica que os periféricos contadores contarão de um em um seguindo a frequencia de clock (FBDIV) calculada pela frequencia do clock externa (SYSCLK) da placa Explorer 16 BR dividida por 8 (DIV_8). Essa linha de códico iniciada por #pragma com o parâmetro FPBDIV (dividor do clcok externo para uso interno do MCU) só tem efeito para programação do MCU. Se o usuário desejar simular esse comportamento no MPLAB SIM deverá configurá-lo para usar esse parâmetro, pois por *default* no MPLAB SIM não o considera.

```
// CONFIGURACAO PARA GRAVACAO
#pragma config FPLLMUL = MUL_16, FPLLIDIV = DIV_2, FPLLODIV = DIV_8, FWDTEN = OFF
#pragma config POSCMOD = HS, FNOSC = PRI, FPBDIV = DIV_8 //(PBCLK is SYSCLK divided by 8,4,2,1)
```

Adicione os defines dos botões e LEDs.

```
// DEFINES

// BOTOES (0:PRESSIONADO 1:LIBERADO)
#define Botao_1 PORTDbits.RD6 //BOTÃO 1
#define Botao_2 PORTDbits.RD7 //BOTÃO 2
#define Botao_3 PORTAbits.RA7 //BOTÃO 3
#define Botao_4 PORTDbits.RD13 //BOTÃO 4
// LED (0:APAGADO 1:ACESO)
#define Led1 LATAbits.LATA0 //LED1
#define Led2 LATAbits.LATA1 //LED2
#define Led3 LATAbits.LATA2 //LED3
#define Led4 LATAbits.LATA3 //LED4
#define Led5 LATAbits.LATA4 //LED5
#define Led6 LATAbits.LATA5 //LED6
#define Led7 LATAbits.LATA6 //LED7
//#define Led8 LATAbits.LATA7 //LED8 // Utilizar como Botao_3 e nao Led_8
```

O *clock* do *timer* é dado pela linha que sai do Prescaler e chega ao registrador que realemnet faza a contagem (o registrador TMR 1). Observe a (Figura 42), retirada do datasheet do MCU. A figura se refere ao periférico contador 1.

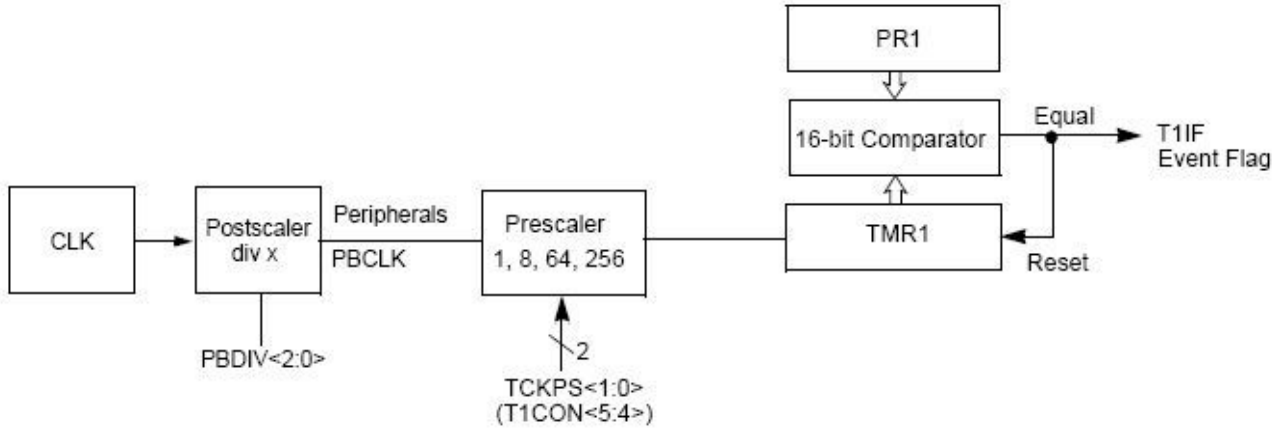

Figura 42 Circuito de Clock do Timer

O *clock* do periférico contador pode ser gerado de várias formas combinado-se a configuração de FPBDIV, o valor do Prescaler, o valor de contagem inicial TMR1, e o valor de parada de contagem PR1. A placa EXPLORER 16 BR contém um oscilador de 8MHz ligado como clock externo SYSCLK ao pino de clock externo do MCU. A frequencia de operação do contador interno ao MCU recebe o sinal externo da placa de 8MHz, passa em seguida pelo circuito divisor PBDIV e por um segundo divisor de frequencias (ou multiplicador de períodos) PRESCALER (divisor de freqüência). Esses três parâmetros são programáveis pelo usuário do MCU. O inverso da freqüência obtida (ou seja, o período) é comparado com o período desejado (PRx – PR1 se usarmos o contador 1 e PR5 se usarmos o contador 5) e quando o contador do *timer* chega a esse valor ele gera uma interrupção, sinalizando que o tempo desejado foi atingido.

O exemplo numérico abaixo ajuda a entender melhor o funcionamento. O exemplo numérico abaixo segue a Equação 1 apresentada anteriormente.

$$SYS_FREQ = 8000000\text{Hz}; (Frequencia\ do\ Sistema)$$
$$PBDIV = 8; (Definido\ no\ modo\ de\ gravação\ como: FPBDIV = \text{DIV}_8)$$
$$PRESCALER = 8 (Definido\ pelo\ usuario)$$
$$Assim\ temos:$$
$$T = \frac{1}{8000000/8/8}$$
$$O\ tempo\ desejado\ para\ o\ timer\ (ex: 1\text{ms})\ é\ dado\ por:$$
$$t = PR * T$$
$$Logo:$$
$$PR = 0.001 * 8000000/8/8 = 125;$$

Inclua a rotina de temporização de 1ms.

```
void delay_1ms_x(unsigned int x)
{
   T5CONbits.TON = 0; // Timer5 desligado
   TMR5 = 0; // Zera o timer para início de contagem

   // Configura o registrador de período
   // PR5 = (Fosc * Tempo)/(FPBDIV * PS)
   // PR5 = (8000000 * 0,001)/(8 * 8) = 125
   PR5 = 125;

   T5CONbits.TCS = 0; // Modo timer (clock interno)
   T5CONbits.TGATE = 0;
```

```
   T5CONbits.TSIDL = 0; // Timer ligado em modo Idle

   // Timer5 Prescaler
   // TCKPS -> Prescaler
   // 0 -> 1:1
   // 1 -> 1:2
   // 2 -> 1:4
   // 3 -> 1:8
   // 4 -> 1:16
   // 5 -> 1:32
   // 6 -> 1:64
   // 7 -> 1:256
   T5CONbits.TCKPS = 3; // Prescaler 1:8

   IFS0bits.T5IF = 0; // Limpa o flag

   T5CONbits.TON = 1; // Timer5 ligado

   while(x != 0)
   {
      while(IFS0bits.T5IF == 0); // Aguarda o tempo de 1 ms
      IFS0bits.T5IF = 0; // Limpa o flag
      x--; // Decrementa x
   }

   T5CONbits.TON = 0; // Desliga o Timer5
}
```

A variável IFS0bits.T5IF no código acima é um *flag* que sinaliza o fim da temporização de 1ms. Ou seja, quando a contagem termina esse *flag* vai para nível lógico alto. Ao resetá-lo (IFS0bits.T5IF=0) o *timer* reinicia sua contagem.

Crie a função **main()**.
Limpe (*reset*) todas as portas.

```
// Reset
LATA = 0;
LATB = 0;
LATC = 0;
LATD = 0;
LATF = 0;
LATG = 0;
```

Configure a direção dos pinos I/O.

```
// Configuração da direção dos pinos de I/O's. (0-Output 1-Input)
DDPCONbits.JTAGEN = 0;

TRISA = 0xFF80; // 1111111110000000 Leds: PORT A0-A6 (Output), Botao_3: PORT A7 (Input)
TRISB = 0xFFFF;
TRISC = 0xFFFF;
TRISD = 0xEFCF; // 1110111111001111 Botoes: PORT D6,D7,D13 (Input)
TRISE = 0xFF00;
TRISF = 0xFFFF;
TRISG = 0xFEBF;
```

Insira o código para fazer com que o LED1 permaneça ligado durante 2 segundos após o botão 1 ser pressionado.

```
while(1) {
   // Atividade 1
   if(!Botao_1) {
      Led1 = 1;
      delay_1ms_x(2000);
      Led1=0;
   } // Fim Atividade 1
}
```

Teste o programa e meça o tempo transcorrido entre o acender e o apagar do LED no MPLAB® SIM. Analise o resultado observado.

Após testar o programa conecte o kit EXPLORER16BR na placa ICD2BR através do cabo RJ12.

Conecte a placa ICD2BR ao computador através da porta USB.

Alimente o kit EXPLORER16BR com a fonte de alimentação.

Compile o programa (**Ctrl+F10**).

Grave o programa (***Program Target Device***).

Teste o programa.

Atividade 1.b

Nesta atividade faremos o Timer1 habilitar uma interrupção, assim, ao invés de chamar a função ***"delay_1_ms_x()"*** o *timer* irá gerar uma interrupção a cada 2 segundos fazendo o led piscar.

Crie um **novo projeto**.

Crie um arquivo **"main.c"**.

Adicione os includes, configure o modo de gravação, adicione os define.

```
// INCLUDES
#include <p32xxxx.h> //include para o PIC32MX360F512L
#include <pblib.h>

// CONFIGURACAO PARA GRAVACAO
#pragma config FPLLMUL = MUL_16, FPLLIDIV = DIV_2, FPLLODIV = DIV_8, FWDTEN = OFF
#pragma config POSCMOD = HS, FNOSC = PRI, FPBDIV = DIV_8 //(PBCLK is SYSCLK divided by 8,4,2,1)

// DEFINES

// BOTOES (0:PRESSIONADO 1:LIBERADO)
#define Botao_1 PORTDbits.RD6 //BOTÃO 1
#define Botao_2 PORTDbits.RD7 //BOTÃO 2
#define Botao_3 PORTAbits.RA7 //BOTÃO 3
#define Botao_4 PORTDbits.RD13 //BOTÃO 4
// LED (0:APAGADO 1:ACESO)
#define Led1 LATAbits.LATA0 //LED1
#define Led2 LATAbits.LATA1 //LED2
#define Led3 LATAbits.LATA2 //LED3
#define Led4 LATAbits.LATA3 //LED4
#define Led5 LATAbits.LATA4 //LED5
#define Led6 LATAbits.LATA5 //LED6
#define Led7 LATAbits.LATA6 //LED7
//#define Led8 LATAbits.LATA7 //LED8 // Utilizar como Botao_3 e nao Led_8
```

Adicione a rotina que será executada quando a interrupção do Timer1 ocorrer.

```
void __ISR(_TIMER_1_VECTOR, ipl2) Timer1Handler(void)
{
   Led1=!Led1;
   IFS0bits.T1IF = 0; // Limpa o flag
}
```

Sempre que ocorre o estouro do contador do Timer1 o *flag* T1IF vai para nível lógico 1, sinalizando a ocorrência de uma interrupção. Ao terminar a rotina de tratamento de interrupção do *Timer* o usuário deve resetá-lo via software, por isso a linha (IFS0bits.T1IF = 0; // Limpa o flag) no código acima.

Inclua uma função para configuração do *Timer*. Perceba que uma função semelhante poderia ter sido utilizada na atividade 1.a.

```
void configura_timer1(){

   T1CONbits.TON = 0; // Timer1 Desligado
   TMR1 = 0; // Zera o timer para início de contagem

   // Timer1 Prescaler
   // TCKPS -> Prescaler
   // 0 -> 1:1
   // 1 -> 1:8
   // 2 -> 1:64
   // 3 -> 1:256
   T1CONbits.TCKPS = 2; // Prescaler 1:64

   // Configura o registrador de período
   // PR1 = (Fosc * Tempo)/(FPBDIV * PS)
   // PR1 = (8000000 * 2)/(8 * 64) = 31250
   PR1 = 31250; // Configura o registrador de período

   T1CONbits.TCS = 0; // Modo timer (clock interno)
   T1CONbits.TGATE = 0;

   IFS0bits.T1IF = 0; // Limpa o flag

   // Configura Inetrrupcao
   IPC1bits.T1IP = 7; //Priority 7 (0,1,...7)
   IPC1bits.T1IS = 0; //SubPriority 0 (0,1,2,3)
   IEC0bits.T1IE = 1; // TMR1 Enable Bit

   // Habilita Interrupção global
   INTEnableSystemMultiVectoredInt();

   T1CONbits.TON = 1; // TMR1 ligado
}
```

Crie a função **Main()**.

Dentro da função **Main()**, limpe (*reset*) todas as portas e configure a direção dos pinos I/O.

```
// Reset
LATA = 0;
LATB = 0;
LATC = 0;
LATD = 0;
LATF = 0;
LATG = 0;

// Configuração da direção dos pinos de I/O's. (0-Output 1-Input)
DDPCONbits.JTAGEN = 0;

TRISA = 0xFF80; // 1111111110000000 Leds: PORT A0-A6 (Output), Botao_3: PORT A7 (Input)
TRISB = 0xFFFF;
TRISC = 0xFFFF;
TRISD = 0xEFCF; // 1110111111001111 Botoes: PORT D6,D7,D13 (Input)
TRISE = 0xFF00;
TRISF = 0xFFFF;
TRISG = 0xFEBF;
```

Configure o Timer1 e insira um *loop while* infinito para que o programa não termine sua execução.

```
   configura_timer1();

   while(1) ;
```

Teste o programa com o MPLAB® SIM. Volte a mensurar o tempo transcorrido entre o acender e o apagar do led. Analise o resultado mensurado. Compare com o que você observou na atividade 1a.

Após testar o programa conecte o kit EXPLORER16BR na placa ICD2BR através do cabo RJ12.

Conecte a placa ICD2BR ao computador através da porta USB.

Alimente o kit EXPLORER16BR com a fonte de alimentação.

Compile o programa (**Ctrl+F10**).

Grave o programa (***Program Target Device***).

Teste o programa.

Atividade 2

Faça o timer1 contar até 8 segundos.

Perceba que se você fizer:

```
// Configura o registrador de período
// PR1 = (Fosc * Tempo)/(FPBDIV * PS)
// PR1 = (8000000 * 8)/(8 * 64) = 125000
PR1 = 125000; // Configura o registrador de período
```

O programa não contará corretamente, pois PR1 é um registrador de 16 bits, logo o valor que lhe será atribuído é 59464. Fazendo com que o timer1 opere com 59464/(8000000/8/64) = 3.8 segundos. (Figura 43)

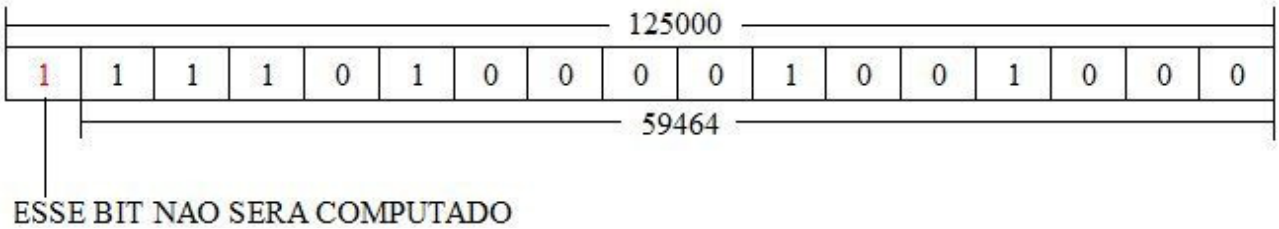

Figura 43 Overload do registrador PR1

Em outras palavras, sendo PR1 um registrador de 16 bits o valor máximo que podemos atribuí-lo é $2^{\wedge 6}$-1 = 65535.

Desta forma, altere o *prescaler* do timer1 (T1CONbits.TCKPS) e o contador (PR1) para obter os 8 segundos desejados.

Atividade 3

O LED1 deve piscar com velocidades que variam de 1 a 5 segundos com incrementos de 1 segundo. Para tanto o botão 1 servirá para incrementar a velocidade enquanto o botão 2 para decrementá-la.

7.4. Aula 4 – Configuração de Interface para Comunicação com Displays de Cristal Líquido (LCDs) alfa numéricos.

Objetivo

Programar recursos de um MCU para se comunicar com *displays LCD* (*Liquid Crystal Display*) alfa-numéricos.

Referências

- Manual do *display* Hantronix (HDM16216H-B), disponível em:
- http://www.hantronix.com/down/char-comm.pdf
- Aula prática 1
- Aula prática 3

Introdução

Os displays LCDs podem ser do tipo alfa-numéricos (exibem caracteres) ou gráficos (pontos, linhas, curvas, etc). Os LCDs alfanuméricos podem ser classificados segundo o número de colunas e linhas de caracteres que ele pode exibir. Nesta pratica iremos trabalhar com LCD alfa-numérico 16x2 (16 colunas e 2 linhas) para exibição de letras e números. (Figura 44)

Figura 44 LCD alfa-numérico 16x2

O display de cristal líquido é um componente importante para interace homem-sistema em sistemas embarcados. Um dispositivo LCD possui um visor onde os caracteres são exibidos. Esse visor pode ter luz de fundo ou não para permitir que o usuário visualize melhor a mensagem que será exibida no mesmo. O visor do LCD é montado em uma placa de circuito impresso com 2 chips soldados no verso da placa. Um chip desses é o controlador do LCD e o outro é uma pequena memória capaz de armazenar uma matriz de caracteres que o usuário utilizará para exibir mensagens na tela. O dispositivo LCD alfa-numérico que usaremos nessa prática possui 8 pinos para dados (D0-D7), 3 pinos para controle e 2 de alimentação. Alguns ainda possuem o controle de luz na parte posterior do visor (*back-light*) para iluminar o LCD em condições de pouca luminosidade.

Os pinos de controle (RS, R/W e E) servem para a comunicação do microcontrolador (MCU) com o controlador do LCD e vice-versa. O pino de controle RS informa se os bits contidos no barramento (D0-D7) são dados (1) ou instrução (0). O pino de controle R/W seleciona entre a escrita (0) na RAM do LCD ou leitura (1) da RAM do LCD. Por fim, o pino de controle E (*enable*) habilita ou não o envio/leitura da informação contida em D0-D7.

Para inicializar o display é necessário enviar os comandos corretos e respeitar o tempo pré-estabelecido pelo fabricante para realizar a próxima tarefa. A Figura 45 abaixo mostra a inicialização do display (Hantronix, Inc 2010) que usaremos nessa prática e logo a seguir um

pequeno código escrito em C, baseado nos dados da parte esquerda da Figura 45, a título de comparação. A parte direita da figura se refere a inicialização do LCD quando programado *nibble* a *nibble* (de 4 em 4 bits)

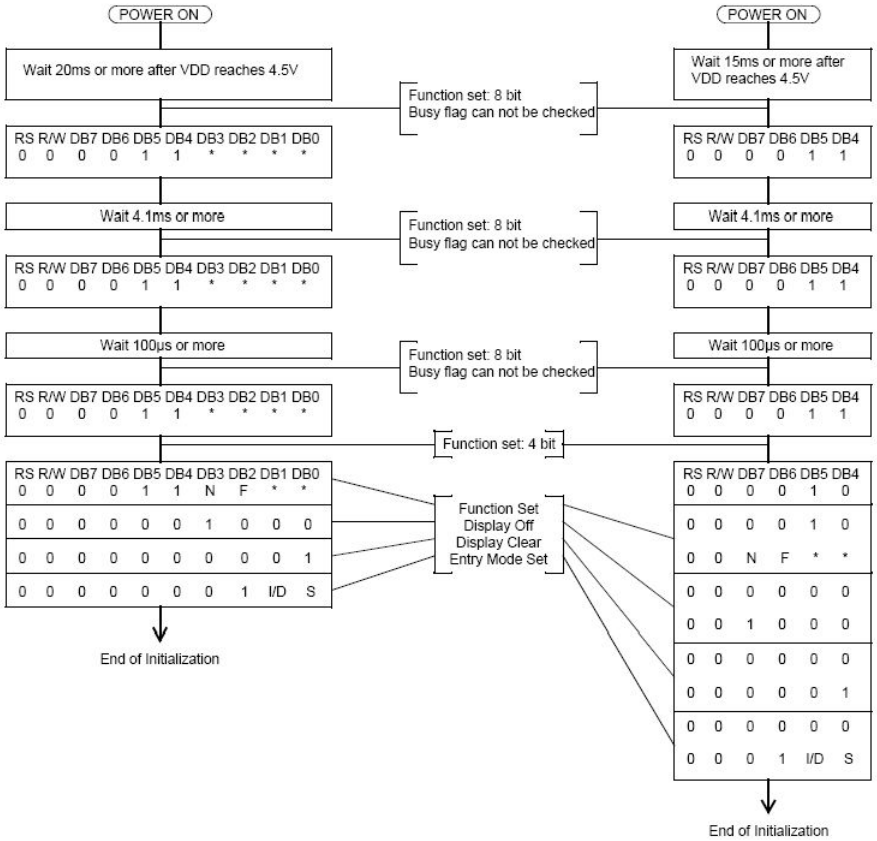

Figura 45 Inicialização do display HDM16216H-B (obtida no datasheet do LSD)

```
void inicializa_LCD(void)
{
   TRIS_E = 0;
   TRIS_RS = 0; // RS=0
   TRIS_RW = 0; // R/W=0

   RW_PIN = 0;
   delay_1ms(20); // Wait 20ms or more after VDD reaches 4.5V
   comando_LCD(0x30); // COMANDO 0x30 : Envia Comando para Inicializar o Display
   delay_1ms(4); // my_timer.c
   comando_LCD(0x30); //COMANDO 0x30 : Envia Comando para Inicializar o Display
   delay_10us(10); // my_timer.c
   comando_LCD(0x30); //COMANDO 0x30 : Envia Comando para Inicializar o Display

   comando_LCD(0x38); //8 BITS DE DADOS, DISPLAY COM 2 LINHAS, CARACTER 7x5 PONTOS
   comando_LCD(0x0c); //DISPLAY ON, SEM CURSOR, SEM BLINK
   comando_LCD(0x01); //LIMPA DISPLAY
   comando_LCD(0x06); //DESLOCAMENTO DO CURSOR P/ DIREITA
}
```

Os comandos a seguir foram extraídos do manual do *display* (Hantronix, Inc 2010).

COMMANDS FOR CHARACTER MODULES

Command	Code										Description	Execution Time
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears the display and returns the cursor to the home position (address 0).	82μs~1.64ms
Return Home	0	0	0	0	0	0	0	0	1	*	Returns the cursor to the home position (address 0). Also returns a shifted display to the home position. DD RAM contents remain unchanged.	40μs~1.64ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets the cursor move direction and enables/disables the display.	40μs
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	Turns the display ON/OFF (D), or the cursor ON/OFF (C), and blink of the character at the cursor position (B).	40μs
Cursor & Display Shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves the cursor and shifts the display without changing the DD RAM contents.	40μs
Function Set	0	0	0	0	1	DL	N$	RE	*	#	Sets the data width (DL), the number of lines in the display (L), and the character font (F).	40μs
Set CG RAM Address	0	0	0	1	A_{CG}						Sets the CG RAM address. CG RAM data can be read or altered after making this setting.	40μs
Set DD RAM Address	0	0	1	A_{DD}							Sets the DD RAM address. Data may be written or read after making this setting.	40μs
Read Busy Flag & Address	0	1	BF	AC							Reads the BUSY flag (BF) indicating that an internal operation is being performed and reads the address counter contents.	1μs
Write Data to CG or DD RAM	1	0	Write Data								Writes data into DD RAM or CG RAM.	46μs
Read Data from CG or DD RAM	1	1	Read Data								Reads data from DD RAM or CG RAM.	46μs
	I/D = 1: Increment S = 1: Accompanies display shift. S/C= 1: Display shift R/L= 1: Shift to the right. DL = 1: 8 bits N = 1: 2 lines RE = 1: Ext. Reg. Ena. BF = 1: Busy # Set to 1 on 24x4 modules $ With KS0072 is Address Mode.						I/D = 0: Decrement S/C = 0: cursor move R/L= 0: Shift to the left. DL = 0: 4 bits N = 0: 1 line F = 0: 5 x 7 dots BF = 0: Can accept data				DD RAM: Display data RAM CG RAM: Character generator RAM A_{CG}: CG RAM Address A_{DD}: DD RAM Address Corresponds to cursor address. AC: Address counter Used for both DD and CG RAM address.	Execution times are typical. If transfers are timed by software and the busy flag is not used, add 10% to the above times.

HANTRONIX Page 48

Figura 46 Comandos do LCD

Outra informação relevante para a correta utilização do *display* é saber o endereço da posição de cada caractere.

Figura 47 Endereço das células do LCD 16x2

Registradores

Nessa prática não há nenhum registrador específico do MCU para programar o LCD além do que os registradores que você usou nas práticas anteriores.

Atividade 1

Como os programas tendem a aumentar muito de tamanho a medida que vamos acrescentando ou programando mais periféricos, por exemplo, nesta aula iremos usar botões, timers e o LCD, o código ficaria muito confuso se programassemos todos eles em um só módulo (arquivo fonte). Por isso, uma boa prática de programação no projeto de microcontroladores será adotada a partir de agora. Trata-se da modularização do código, onde criaremos arquivos fonte com extensão (.c) e arquivos de cabeçalho (*headers*) com extensão (.h) a fim de tornar o código mais legível e organizado.

Primeiramente crie um arquivo chamado *"init_sys.c"* e *"init_sys.h"*. Salve-os na pasta do seu projeto. (*"File New"*, *"File Save as..."*). Nesse arquivo programaremos a inicialização do sistema. Inclua os arquivos no seu projeto: Na aba *"Source Files"* da área de trabalho clique com o botão direito do mouse e escolha *"Add Files..."*, na janela que se abre selecione o arquivo *"init_sys.c"*. Na aba *"Header Files"* da área de trabalho clique com o botão direito do mouse e escolha *"Add Files.."*, na janela que se abre selecione o arquivo *"init_sys.h"*.

No arquivo *"init_sys.h"* insira o seguinte código:

```
#ifndef INIT_SYS_H
#define INIT_SYS_H
#include <p32xxxx.h> //include para o PIC32MX360F512
void init_sys(void);
#endif
```

No arquivo *"init_sys.c"* Insira o seguinte código:

```
#include "init_sys.h"

void init_sys(){
   // Reset
   LATA = 0;
   LATB = 0;
   LATC = 0;
   LATD = 0;
   LATF = 0;
   LATG = 0;

   // Configuração da direção dos pinos de I/O's. (0-Output 1-Input)
   DDPCONbits.JTAGEN = 0;

   TRISA = 0xFF80; // 1111111110000000 Leds: PORT A0-A6 (Output), Botao_3: PORT A7 (Input)
   TRISB = 0xFFFF;
   TRISC = 0xFFFF;
   TRISD = 0xEFCF; // 1110111111001111 Botoes: PORT D6,D7,D13 (Input)
   TRISE = 0xFF00;
   TRISF = 0xFFFF;
   TRISG = 0xFEBF;
}
```

Inclua a referência do arquivo em "main.c".

```
#include "init_sys.h"
```

Isso fará com que as funções e variáveis do arquivo *"init_sys.h"* possam ser interpretadas dentro do arquivo *"main.c"*.

Agora faça o mesmo para *"my_timer.c"* e *"my_timer.h"*. Esses arquivos conterão as temporizações necessárias para o MCU se comunicar corretamente com a controladora do LCD.

Em *"my_timer.c"* inclua o seguinte código:

```
#include "init_sys.h"

void delay (int PR, unsigned int x)
{

   T5CONbits.TON = 0; // TMR OFF
   TMR5 = 0; // ZERA CONTADOR

   PR5 = PR;
   T5CONbits.TCKPS = 3; // Prescaler 1:8

   T5CONbits.TCS = 0; // CLK INTERNO
   T5CONbits.TGATE = 0;

   IFS0bits.T5IF = 0; // Limpa flag
   T5CONbits.TON = 1; // TMR ON

   while(x != 0) { // Temporizacao
      while(IFS0bits.T5IF == 0);
      IFS0bits.T5IF = 0;
      x--;
   }

   T5CONbits.TON = 0; // TMR OFF
}

// Delay = 1ms*X
void delay_1ms(unsigned int x)
{
// Periodo TMR : (Fosc * Tempo)/(FPBDIV * PS)
// PR5 = (8000000 * 0,001)/(2 * 8) = 500
PR5 = 500;
delay(500,x);
}


// Delay = 10us*X
void delay_10us(unsigned int x)
{
   // Periodo TMR : (Fosc * Tempo)/(FPBDIV * PS)
   // PR5 = (8000000 * 0.000010)/(2 * 8) = 5
   PR5 = 5;
   delay(5,x);
}
```

Esse código contem o cálculo do período do *timer* e do *prescaler* necessários para se obter o tempo desejado. Caso você não entenda o código acima, retorne a aula 3 e estude como configurar os registradores necessários ao uso de *timers* e contadores.

Em "my_timer.h" insira o código:

```
#ifndef MY_TIMER_H
#define MY_TIMER_H

#include <p32xxxx.h> //include para o PIC32MX360F512

void delay(int, unsigned int); //ATRASO
void delay_1ms(unsigned int); //ATRASO MÚLTIPLO DE 1MS
void delay_10us(unsigned int); //ATRASO MÚLTIPLO DE 10US

#endif
```

Por fim crie os arquivos *"my_lcd.c"* e *"my_lcd.h"*. Esses arquivos conterão as funções especificas do *lcd*.

Em *"my_lcd.c"* inclua o seguinte código:

```
#include "my_lcd.h"

// LCD Init
void inicializa_LCD(void)
{

    TRIS_E = 0;
    TRIS_RS = 0; // RS=0
    TRIS_RW = 0; // R/W=0

    RW_PIN = 0;
    delay_1ms(100); // Wait 20ms or more after VDD reaches 4.5V
    comando_LCD(0x30); // COMANDO 0x30 : Envia Comando para Inicializar o Display

    delay_1ms(4);
    comando_LCD(0x30); //COMANDO 0x30 : Envia Comando para Inicializar o Display

    delay_10us(10);
    comando_LCD(0x30); //COMANDO 0x30 : Envia Comando para Inicializar o Display

    comando_LCD(0x38); //COMANDO DISPLAY DE 8 VIAS, 2 LINHAS, E CARACTER 7X5
    comando_LCD(0x0c); //DISPLAY ON, SEM CURSOR, SEM BLINK
    comando_LCD(0x01); //LIMPA DISPLAY
    comando_LCD(0x06); //DESLOCAMENTO DO CURSOR P/ DIREITA

}

// Limpa LCD
void limpar_LCD(void)
{

    comando_LCD(0x01); //ENVIA COMANDO 0x01
    delay_1ms(2); //DELAY DE 2ms

}

// Envia Dado
void dado_LCD(unsigned char dado)
{

    LATE = (TRISE&0xFFFFFF00);     // Configura pinos LCD como saida
    RS_PIN = 1; // SELECIONA PARA DADOS
    LATE = (LATE&0xFFFFFF00)|dado;     // Escreve Dados na Port
    E_PIN = 1; // Envia Dado
    delay_10us(1);
    E_PIN = 0;
    delay_10us(4);
    LATE = (TRISE|0x000000FF);     // Configura pinos LCD como entrada

}

// Envia Comando
void comando_LCD(unsigned char dado)
{

    LATE = (TRISE&0xFFFFFF00);     // Configura pino como saída
    RS_PIN = 0; // SELECIONA PARA DADOS
    LATE = (LATE&0xFFFFFF00)|dado;     // Escreve Dados na Port
    E_PIN = 1; // Envia Dado
    delay_10us(1);
    E_PIN = 0;
    delay_10us(4);
    LATE = (TRISE|0x000000FF);    // Configura pinos LCD como entrada

}

// Escreve Lcd
void escreve_frase_LCD(const char *frase)
{
    do
    {
        dado_LCD(*frase);
    }while(*++frase);
}
```

Em *"my_lcd.h"* inclua o seguinte código:

```
#ifndef MY_LCD_H
#define MY_LCD_H

#include <p32xxxx.h> //include para o PIC32MX360F512

// PINOS LCD (CONTROLE E DADOS)
#define DATA_PIN_7 LATEbits.LATE7
#define DATA_PIN_6 LATEbits.LATE6
#define DATA_PIN_5 LATEbits.LATE5
#define DATA_PIN_4 LATEbits.LATE4
#define DATA_PIN_3 LATEbits.LATE3
#define DATA_PIN_2 LATEbits.LATE2
#define DATA_PIN_1 LATEbits.LATE1
#define DATA_PIN_0 LATEbits.LATE0

#define TRIS_DATA_PIN_7 TRISEbits.TRISE7
#define TRIS_DATA_PIN_6 TRISEbits.TRISE6
#define TRIS_DATA_PIN_5 TRISEbits.TRISE5
#define TRIS_DATA_PIN_4 TRISEbits.TRISE4
#define TRIS_DATA_PIN_3 TRISEbits.TRISE3
#define TRIS_DATA_PIN_2 TRISEbits.TRISE2
#define TRIS_DATA_PIN_1 TRISEbits.TRISE1
#define TRIS_DATA_PIN_0 TRISEbits.TRISE0

#define E_PIN LATDbits.LATD4
#define RS_PIN LATBbits.LATB15
#define RW_PIN LATDbits.LATD5

#define TRIS_E TRISDbits.TRISD4
#define TRIS_RS TRISBbits.TRISB15
#define TRIS_RW TRISDbits.TRISD5

void inicializa_LCD(void);        //INCIALIZA LCD
void limpar_LCD(void);            //LIMPA LCD
void comando_LCD(unsigned char);        //ENVIA COMANDO AO LCD
void dado_LCD(unsigned char);            //ESCREVE BYTE NO LCD
void escreve_frase_LCD(const char *);     //ESCREVE FRASE NO LCD

#endif
```

Em "*main.c*" insira o código para a atividade 1.

```
// INCLUDES
#include <p32xxxx.h> //include para o PIC32MX360F512
#include <stdio.h>
#include <stdlib.h>
#include "..\\header/init_sys.h"
#include "..\\header/my_timer.h"

// CONFIGURACAO PARA GRAVACAO
#pragma config FPLLMUL = MUL_16, FPLLIDIV = DIV_2, FPLLODIV = DIV_8, FWDTEN = OFF
#pragma config POSCMOD = HS, FNOSC = PRI, FPBDIV = DIV_2



// DEFINES
// BOTOES (0:PRESSIONADO 1:LIBERADO)
#define Botao_1 PORTDbits.RD6 //BOTÃO 1
#define Botao_2 PORTDbits.RD7 //BOTÃO 2
#define Botao_3 PORTAbits.RA7 //BOTÃO 3
#define Botao_4 PORTDbits.RD13 //BOTÃO 4
// LED (0:APAGADO 1:ACESO)
#define Led1 LATAbits.LATA0 //LED1
#define Led2 LATAbits.LATA1 //LED2
#define Led3 LATAbits.LATA2 //LED3
#define Led4 LATAbits.LATA3 //LED4
#define Led5 LATAbits.LATA4 //LED5
#define Led6 LATAbits.LATA5 //LED6
#define Led7 LATAbits.LATA6 //LED7
```

```
//#define Led8 LATAbits.LATA7 //LED8 // Utilizar como Botao_3 e nao Led_8

void aula4_atividade1 (void){

inicializa_LCD(); //INICIALIZA LCD
limpar_LCD(); //LIMPA O LCD

while(1)
{
   if(!Botao_1)
   {
      limpar_LCD(); // LIMPA LCD
      comando_LCD(0x80); // POSICIONA CURSOR NA LINHA 0 COLUNA 0
      escreve_frase_LCD("TECLA 1"); // ESCREVE MENSAGEM
      while(!Botao_1);
      limpar_LCD();
   }
   else if(!Botao_2)
   {
      limpar_LCD();
      comando_LCD(0xC0); //POSICIONA CURSOR NA LINHA 1 COLUNA 0
      escreve_frase_LCD("TECLA 2");
      while(!Botao_2);
      limpar_LCD();
   }
   else if(!Botao_3)
   {
      limpar_LCD();
      comando_LCD(0x89); //POSICIONA CURSOR NA LINHA 0 COLUNA 9
      escreve_frase_LCD("TECLA 3");
      while(!Botao_3);
      limpar_LCD();
   }
   else if(!Botao_4)
   {
      limpar_LCD();
      comando_LCD(0xC9); //POSICIONA CURSOR NA LINHA 1 COLUNA 9
      escreve_frase_LCD("TECLA 4");
      while(!Botao_4);
      limpar_LCD();
   }
   else
   {
      comando_LCD(0x80); //POSICIONA CURSOR NA LINHA 0 COLUNA 0
      escreve_frase_LCD("UFMG - AULA4");
      comando_LCD(0xC0); //POSICIONA CURSOR NA LINHA 1 COLUNA 0
      escreve_frase_LCD("Atividade 1");
      LATA = 0;
   }
}

}




int main(void)
{
      init_sys(); // Inicializa o sistema
      aula4_atividade1(); // Chamada da funcao para atividade 1
}
```

Conecte o kit EXPLORER16BR na placa ICD2BR através do cabo RJ12.
Conecte a placa ICD2BR ao computador através da porta USB.
Alimente o kit EXPLORER16BR com a fonte de alimentação.
Compile o programa (Ctrl+F10).
Grave o programa (*Program Target Device*).
Teste o programa.

Atividade 2

Programe o MCU para escrever a seguinte frase (centralizada) no LCD:

		U	F	M	G		-		A	U	L	A	4		
		A	T	I	V	I	D	A	D	E		#	1		

Quando o botão 1 for pressionado o LCD deverá exibir a seguinte mensagem:

											T	e	c	l	a
P	r	e	s	s	i	o	n	a	d	a					

Atividade 3

Faça um cronômetro de segundos e minutos com contagem de tempo progressiva. Faça com que o usuário possa definir um valor qualquer de entrada para os minutos e segundos maior ou igual a zero.

Dica: Utilize uma temporização de 1 segundo, haja visto que essa é a menor unidade de medida temporal necessária para a atividade.

7.5. Aula 5 – Configuração de Conversores Analógicos/Digitais (Conversores A/D)

Objetivo

Programação e uso de conversores analógico-digitais para aquisição de sinais analógicos.

Referências

- *Datasheet* PIC32MX3XX-4XX disponível no CD1 em ***"Datasheets >> PIC32"***. Capítulo 22.0: *Analog-Digital Converter*
- Aula prática 1
- Aula prática 2
- Aula pratica 3
- Aula pratica 4

Introdução

A maior parte dos fenômenos físicos da natureza são representados por grandezas físicas que descrevem quantitativamente a sua variação de forma contínua em função do tempo. A esses sinais denominamos sinais contínuos ou sinais analógicos.

Um sinal discreto é uma série temporal que consiste de uma sequência de quantidades, uma função sobre o domínio de inteiros discretos. Cada valor da sequência é chamado de amostra. Diferente do sinal contínuo, um sinal discreto não é uma função de um argumento contínuo. Entretanto, a função pode ter sido obtida através da amostragem de um sinal contínuo. Uma forma de se obter um sinal discreto é a aquisição de valores de um sinal analógico numa determinada taxa de tempo, a esse processo chamamos de amostragem de sinal.

Os sinais analógicos são muitas vezes processados por circuitos digitais, por exemplo, por um microcontrolador ou por um microcomputador. Para processar sinais analógicos usando circuitos digitais, deve-se efetuar uma conversão para essa última forma, a digital. Tal conversão é efetuada por um Conversor Analógico-Digital. O conversor analógico-digital (frequentemente abreviado por conversor A/D) é um dispositivo eletrônico capaz de gerar uma representação digital (discreta) de uma grandeza analógica (contínua).

Por exemplo, um conversor A/D de 10 bits, preparado para um sinal de entrada analógica de tensão variável de 0V a 5V pode gerar números binários de 0 (0000000000) a 1023 (1111111111) (ou seja, capturar 1024 pontos do sinal), dependendo do sinal de entrada. Se o sinal de entrada do suposto conversor A/D estiver em 2,5V, o valor binário gerado será 511 ou 512.

Dessa forma, conversores analógico-digitais são importantes para se fazer a amostragem de sinais reais afim de discretizá-los para serem manipulados pelos sistemas digitais (ex: microcontroladores).

Existem ainda os conversores digitais-analógicos os quais fazem a conversão de sinais discretos no tempo, provenientes do controlador, para sinais contínuos afim de gerar uma referência/comando para os sistemas reais. A interface entre sistemas digitais e analógicos está exemplificada na Figura 48.

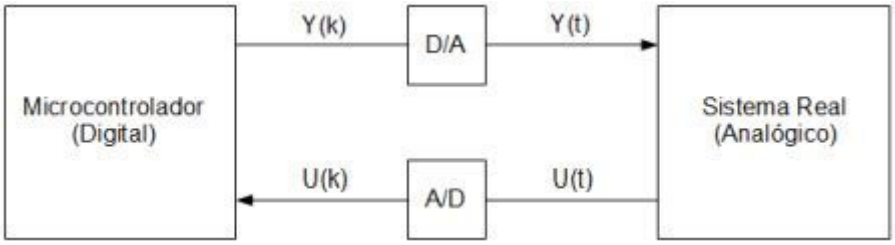

Figura 48 Conversão A/D - D/A

Nesta prática configuraremos (programaremos) o conversor analógico-digital (A/D) do MCU para leitura do nível de tensão analógico externo.

Os conversores analógico-digitais se diferenciam basicamente por 4 aspectos, a saber: método usado na conversão, resolução, frequência de Amostragem e forma de apresentação do resultado da conversão.

Os métodos de conversão A/D mais comuns a saber são: método por realimentado, método paralelo (*flash*), método sigma-delta, método por aproximações sucessivas, dentre outras. O PIC32MX360F512, por exemplo, utiliza o método de aproximações sucessivas.

A resolução diz respeito ao numero de níveis que podem ser representados pelo conversor. Por exemplo, o PIC32MX360F512 possui um conversor A/D que usa 10 bits para conversão. Portanto ele por apresentar até 2^{10} = 1024 níveis diferentes do sinal analógico amostrado.

A frequência de amostragem relaciona-se com quão rápido o dispositivo irá fazer a aquisição do sinal. Essa frequência deve ser maior que duas vezes a maior frequência contida do sinal de entrada. Por exemplo, se quisermos fazer a leitura de um sinal cuja frequência máxima seja 2 Khz, a frequência de amostragem deve ser superior a 4 Khz. Se aumentarmos muito a frequência o custo computacional também irá aumentar.

O último aspecto, apresentação do resultado da conversão, determina quantos bits serão utilizados para a conversão (para o conversor A/D do PIC32MX360F512 podemos escolher entre 16 ou 32 bits), se o resultado será sinalizado ou não e ainda se ele será armazenado nos bits mais significativos ou nos bits menos significativos do registrador.

No PIC32MX360F512 os canais de conversão AD são pinos representados pela sigla ANx, onde x representa o número do canal para conversão (0-15). É importante ressaltar que esses pinos podem ser utilizados para outras tarefas diferentes que não uma conversão AD. Por exemplo, o pino 25 do PIC32MX360F512 denominado ***"PGD1/EMUD1/AN0/CN2/RB0"*** pode ser utilizado como: canal de comunicação para o *debugger in circuit* (PGD1); canal de comunicação para o emulador *in circuit* (EMUD1); conversor analógico-digital (AD0), interrupção (CN2) ou ainda simplesmentes como um pino para comunicação digital (RB0).

Compreendido que os canais de conversão A/D estão associados a pinos específicos do PIC, que a quantidade e localização dos conversores variam conforme o microcontrolador utilizado e, tendo em vista que o PIC32MX360F512 trabalha com o método de aproximações sucessivas e possui 10 bits para conversão passaremos para a explicação de como é feita a conversão do sinal.

Quando o sinal analógico é inserido no pino correspondente do conversor existe um circuito denominado *Sample and Hold* (amostra e congela). Em um primeiro momento o capacitor se carrega com a tensão do sinal de entrada, esse instante é denominado *Acquisition Time* (período de aquisição). Posteriormente o capacitor é chaveado para se realizar a leitura do valor obtido no

primeiro passo, esse instante é denominado *A/D Conversion Time* (período de conversão). (Veja Figura 49 e Figura 50) É nesse instante que o método de conversão por aproximações sucessivas é utilizado.

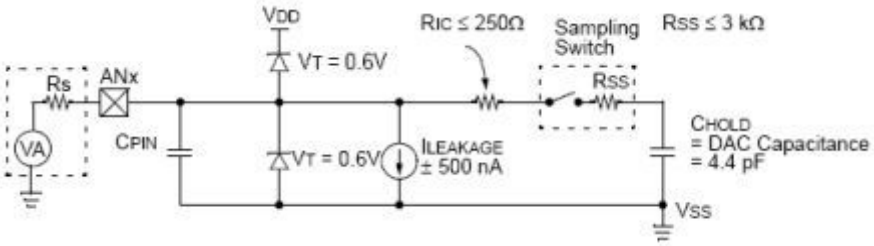

Figura 49 Circuito de amostragem do sinal

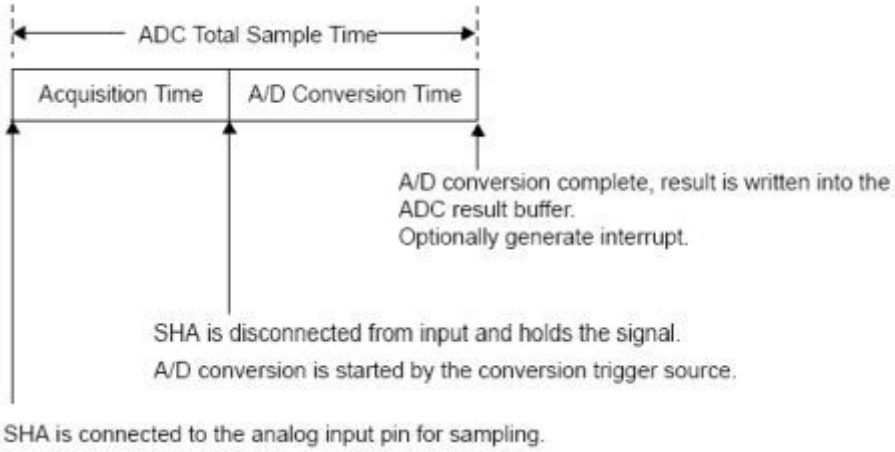

Figura 50 Sequencia de amostragem e conversão do ADC

Para melhor entendimento imagine uma tensão que varie de 0 a 3,3 Volts. Se tivermos um conversor A/D de 1 bit só teremos dois estados, logo só seremos capazes de identificar 0V ou 3,3V com cada passo equivalente a 3,3V. Agora, se tivermos um conversor de 2 bits teremos 4 estados, logo, seremos capazes de representar os números 0V, 1,1V, 2,2V e 3,3V com cada passo equivalente a 1,1V. Se tivermos um conversor de 3 bits teremos 8 estados e podemos representar os números 0V, 0,47V, 0,94V, 1,41V, 1,88V, 2,35V, 2,82V e 3,29V com cada passo equivalente a 0,47V. Ou seja, quanto maior o número de bits do conversor menor o passo e portanto maior a possibilidade de números que podem ser representados corretamente em relação à tensão de referência. No caso do conversor A/D existente no PIC32MX360F512 temos 10 bits de resolução, portanto 2^{10}=1024 passos. O valor de cada passo é dado por: $V_{REF}/2^{10} = V_{REF}/1024$. Nesta pratica iremos utilizar V_{REF} = 3,3V, que é a tensão de referencia interna do PIC32MX360F512[11].

Em uma representação binária o bit mais significativo (*MSB – Most Significant Bit*) representa metade do valor máximo representável. O segundo bit mais significativo representa metade da metade, ou seja, um quarto do valor máximo representável. O terceiro *bit* mais significativo representa um oitavo e assim sucessivamente. Como exemplo considere um número de 4 bits. O valor máximo que podemos representar é 2^4=16, quando todos os bits estão em '1'. Agora, se somente o *bit* mais significativo estiver em '1' o resultado será '1000' o que equivale a 8 (metade do valor máximo). Se somente o segundo bit mais significativo estiver em '1' o resultado será '0100' o que equivale a 4 (um quarto do valor máximo).

[11] Embora seja utilizado nessa pratica a tensão de referencia interna é possível utilizar referencias externas de tensão através da configuração correta do módulo ADC.

O método de conversão por aproximações sucessivas realiza a conversão baseado nesse principio. Primeiro ele verifica se a tensão a ser convertida é maior ou menor que metade da tensão de referência (no nosso caso, 3,3V). Caso a tensão seja maior o MSB recebe '1', caso a tensão seja menor o MSB recebe '0'. Depois o conversor considera um quarto da tensão de referência. Caso o MSB tenha sido '1' o conversor compara a tensão de entrada com metade mais um quarto da tensão de referência. Caso o MSB tenha sido '0' o conversor compara a tensão de entrada com metade menos um quarto da tensão de referência. Este processo de comparação prossegue até o décimo *bit* ser preenchido, momento no qual a conversão é finalizada.

Como exemplo considere uma tensão de entrada de 2,0365V e a tensão de referência de 3,3V.

1º - o conversor verifica de a tensão de entrada é superior ou inferior a 3,3/2=1,65V. No caso ela é superior. Dessa forma o MSB recebe o valor '1'.

2º - o conversor verifica se a tensão de referência é superior ou inferior a (3,3/2)+(3,3/4)=2,475V. No caso ela é inferior. Dessa forma o segundo *bit* recebe o valor '0'.

3º - o conversor verifica se a tensão de referência é superior ou inferior a (3,3/2)+(3,3/4)-(3,3/8)=2,0625V. No caso ela é inferior. Dessa forma o terceiro *bit* recebe o valor '0'.

4º - o conversor verifica se a tensão de referência é superior ou inferior a (3,3/2)+(3,3/4)-(3,3/8)-(3,3/16)=1,856V. No caso ela é superior. Dessa forma o quarto *bit* recebe o valor '1'.

5º - o conversor verifica se a tensão de referência é superior ou inferior a (3,3/2)+(3,3/4)-(3,3/8)-(3,3/16)+(3,3/32)=1,959V. No caso ela é superior. Dessa forma o quinto *bit* recebe o valor '1'.

6º - o conversor verifica se a tensão de referência é superior ou inferior a (3,3/2)+(3,3/4)-(3,3/8)-(3,3/16)+(3,3/32)+(3,3/64)=2,011V. No caso ela é superior. Dessa forma o sexto *bit* recebe o valor '1'.

7º - o conversor verifica se a tensão de referência é superior ou inferior a (3,3/2)+(3,3/4)-(3,3/8)-(3,3/16)+(3,3/32)+(3,3/64)+(3,3/128)=2,0367V. No caso ela é inferior. Dessa forma o sétimo *bit* recebe o valor '0'.

8º - o conversor verifica se a tensão de referência é superior ou inferior a (3,3/2)+(3,3/4)-(3,3/8)-(3,3/16)+(3,3/64)+(3,3/128)-(3,3/256)=2,0238V. No caso ela é superior. Dessa forma o oitavo *bit* recebe o valor '0'.

9º - o conversor verifica se a tensão de referência é superior ou inferior a (3,3/2)+(3,3/4)-(3,3/8)-(3,3/16)+(3,3/64)+(3,3/128)-(3,3/256)+(3,3/512)=2,0303V. No caso ela é superior. Dessa forma o nono *bit* recebe o valor '1'.

10º - o conversor verifica se a tensão de referência é superior ou inferior a (3,3/2)+(3,3/4)-(3,3/8)-(3,3/16)+(3,3/64)+(3,3/128)-(3,3/256)+(3,3/512)+(3,3/1024)=2,0334V. No caso ela é superior. Dessa forma o décimo *bit* (LSB – *Less Significant Bit*) recebe o valor '1'.

Assim o resultado binário da conversão é : 1001110011

Perceba que são necessárias 10 iterações para se converter os 10 bits que representam o resultado da conversão. Em geral, para um conversor de n bits são necessárias n iterações com o método de aproximações sucessivas.

Uma maneira simples de transformar o valor da conversão (valor entre 0 e 1023, no nosso caso) em um valor real (valor entre 0 e V_{REF}) é multiplicar o resultado da conversão pela resolução do conversor. Para o nosso caso teremos:

$$V_{REF} = 3{,}3\,V$$
$$N = 10\,Bits$$
$$niveis = 2^{10} = 1024$$
$$Resolução = 3{,}3/1024 = 0{,}00322265625$$
$$Logo:$$
$$valor_{real} = resultado_{ad} * 0.00322265625.$$

Registradores

O PIC32MX360F512L possui 16 pinos (RB0 – RB15) que podem ser programados como entrada analógica (AN0 – AN15) para conversão analógico-digital.

Vale ressaltar que apesar de o PIC possuir 16 canais configuráveis como entrada analógica, internamente existe somente um conversor. Dessa forma as entradas devem ser multiplexadas de forma a se fazer a leitura de cada canal individualmente.

Os registradores envolvidos na programação do modulo A/D são:

TRISx – Configura o pino como entrada. (ver 7.1. Aula 1 – Configuração de Pinos de Entrada e Saída)

AD1PCFG – Esse registrador é utilizado para informar se o pino correspondente será setado como digital ou analógico, uma vez que ele já tenha sido configurado como entrada através do registrador TRISx. Nível lógico "1" implica que o pino correspondente será digital, nível lógico "0" implica em pino analógico.
Ex.: AD1PCFG = 0xFFDF; // 1111 1111 1101 1111
A configuração de AD1PCFG no exemplo acima indica que somente o pino RB5 funcionará como entrada analógica (AN5).

AD1CON1 (*ADC Control Register 1*) – Esse registrador e responsável por:

Habilitar (nível lógico "1") ou desabilitar (nível lógico "0") o conversor A/D através do bit AD1CON1bits.ADON.

Configurar o modo em que o resultado da conversão será exibido (sinalizado ou não sinalizado; 16 ou 32 bits) através do bit AD1CON1bits.FORM.

Configurar, através do bit AD1CON1bits.SSRC, quando a conversão devera iniciar após o sinal ser amostrado.

Habilitar (nível lógico "1") ou desabilitar (nível lógico "0") a amostragem automática. Caso a amostragem automática esteja habilitada a amostragem se inicia logo após a última conversão, caso contrário, ela se iniciará somente após setar o bit de amostragem AD1CON1bits.SAMP.

AD1CON2 (*ADC Control Register 2*) – Esse registrador e responsável por:

Selecionar a referencia de tensão (referencia externa ou interna) para o conversor através do bit AD1CON2bits.VCFG.

Configurar, através do bit AD1CON2bits.SMPI, o numero de conversões que devem ser feitas (1 a 16) antes que a interrupção seja gerada.

Configurar o modo de armazenamento do resultado (dois buffers de 8bits ou um buffer de 16 bits) através do bit AD1CON2bits.BUFM.

O PIC possui dois multiplexadores (MUXA e MUXB) ligados ao conversor. O bit AD1CON2bits.ALTS seleciona se o PIC deve alternar entre os multiplexadores (nível lógico "1") ou deve utilizar sempre o MUXA (nível lógico "0").

AD1CON3 (*ADC Control Register 3*) – Esse registrador e responsável por:

Selecionar a fonte de clock para o modulo A/D através do bit AD1CON3bits.ADRC. Nível lógico "1" indica que o clock e interno ao modulo A/D. Nível lógico "0" indica que o clock e proveniente do barramento de clock do PIC.

Selecionar o tempo de conversão para o modulo A/D através do bit AD1CON3bits.SAMC com sendo um múltiplo do período do clock do barramento do PIC.

Selecionar a base de tempo para o modulo A/D através do bit AD1CON3bits.ADCS.

AD1CHS (ADC Input Select Register) – Este registrador e responsável por selecionar os sinais a serem ligados na entrada positiva e negativa dos MUXA e MUXB.
Ex.: AD1CHSbits.CH0SA = 5; // Conecta o canal 5 (RB5/AN5) na entrada positiva do MUXA.
AD1CHSbits.CH0NA = 0; // Conecta a referencia negativa (selecionada através do bit VCFG do registrador AD1CON2) na entrada negativa do MUXA.

Atividade 1

Após criar um novo projeto no MPLAB inclua os arquivos ***"init_sys.c"***, ***"init_sys.h"***, ***"my_lcd.c"***, ***"my_lcd.h"***, ***"my_timer.c"*** e ***"my_timer.h"*** utilizados na prática anterior. Lembre-se de fazer uma cópia dos arquivos para a pasta do projeto que você acabou de criar. Pois se você fizer ***"Source Files-> Add Files"*** uma referência para o arquivo da aula anterior sera criada e não o arquivo propriamente dito. Para melhor esclarecimento veja: Anexo I : Adicionando arquivos durante a criação do projeto e Anexo II : Adicionando arquivos depois da criação do projeto.

Agora crie um arquivo chamado ***"my_adc.c"*** e ***"my_adc.h"***. Salve (***"File New"***, ***"File Save as..."***) e inclua os arquivos no seu projeto: Na aba ***"Source Files"*** da área de trabalho clique com o botão direito do mouse e escolha ***"Add Files..."***, na janela que se abre selecione o arquivo ***"my_adc.c"***. Na aba ***"Header Files"*** da área de trabalho clique com o botão direito do mouse e escolha ***"Add Files.."***, na janela que se abre selecione o arquivo ***"my_adc.h"***.

No arquivo ***"my_adc.h"*** insira o seguinte código:

```
#ifndef MY_ADC_H
#define MY_ADC_H

#include <p32xxxx.h> //include para o PIC32MX360F512

#define SYS_FREQ (80000000L)

void inicializa_adc(void);

#endif
```

No arquivo ***"my_adc.c"*** insira o seguinte código:

```
1    #include "my_adc.h"
2
3     void inicializa_adc(){
4         AD1CON1bits.ADON = 0; // Desabilita módulo AD
5
6         AD1CHSbits.CH0SA = 5; // Conecta canal 5 (RB5/AN5) na entrada positiva do MUXA
7         AD1CHSbits.CH0NA = 0; // Conecta a referencia negativa (Vr-) na entrada negativa do MUXA
8
9         AD1CON1bits.ADSIDL = 1; // Conversor AD funciona em modo Idle
10        AD1CON1bits.FORM = 0; // Resultado da conversão em 16bits
11        AD1CON1bits.SSRC = 7; // Conversão automatica apos amostragem
12        AD1CON1bits.ASAM = 1; // Amostragem automática Habilitada
13
14        AD1CON2bits.VCFG = 0; // Referencia interna de tensão (AVdd e AVss)
15        AD1CON2bits.CSCNA = 0; // Auto scan desabilitado
16        AD1CON2bits.SMPI = 0; // Interrupção a cada conversão
17        AD1CON2bits.BUFM = 0; // Resultado armazenado em 1 buffer de 16bits
18        AD1CON2bits.ALTS = 0; // Nao alterna MUX A com MUXB
19
20        AD1CON3bits.ADRC = 0; // Clock proveniente do clock de perifericos do PIC (TPB)
21        AD1CON3bits.SAMC = 31; // Tempo de amostragem 31 Tad
22        AD1CON3bits.ADCS = 128; // Tad = 128 x TPB
23
24        AD1PCFG = 0xFFDF; // RB5/AN5 como entrada analógica
25
26        AD1CSSL = 0x0000; // Auto scan desabilitado
27
28        AD1CON1bits.ADON = 1; // Habilita o módulo AD
29    }
30
```

Para entender a configuração do código acima considere a Figura 51. Ela mostra que a entrada RB5/AN5 do PIC está conectada a um divisor de tensão montado na placa Explorer 16 BR que será utilizado como entrada do nosso sinal analógico. Por isso a entrada RB5/AN5 deve ser

configurada como entrada analógica conforme mostrado na linha 24 do código acima (AD1PCFG = 0xFFDF, 1 indica entrada digital e 0 indica entrada analógica).

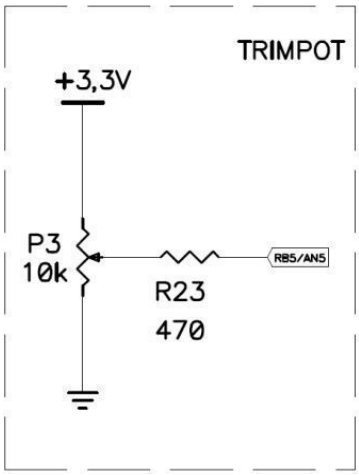

Figura 51 Esquema elétrico do trimpot do kit EXPLORER16BR para utilizacão do conversor AD

O diagrama em blocos do conversor AD pode ser visto na Figura 52. Ao se fazer AD1CON2bits.VCFG=0 estamos tomando os sinais de alimentação do PIC (AVDD=3.3V e AVSS=0V) como referência de tensão. Caso seja necessário podemos utilizar referências externas (VREF+ e VREF-) variando-se a configuração desse bit.

Ao se fazer AD1CHSbits.CH0SA=5 e AD1CHSbits.CH0NA=0 estamos conectando, respectivamente, o pino RB5/AN5 (ou canal 5) na entrada positiva e a referência negativa de tensão (AVSS) na entrada negativa do circuito *sample and hold.*

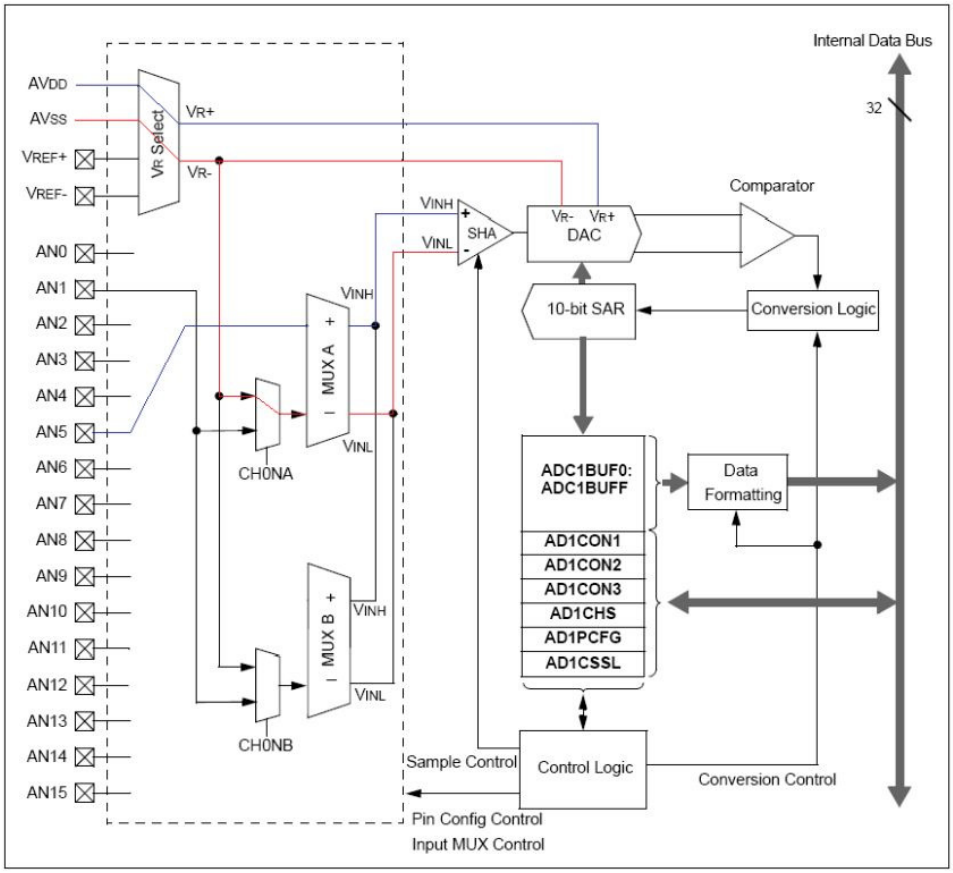

Figura 52 Diagrama em blocos módulo AD

A configuração AD1CON1bits.FORM = 0 implica que o resultado da conversão será uma palavra de 16 *bits* não sinalizada conforme mostra a Figura 53. Como o resultado da conversão possui 10 *bits* este pode ser armazenado em um *buffer* de 16 *bits* (AD1CON2bits.BUFM = 0) ou em dois de 8 *bits*.

REGISTER 22-1: AD1CON1: ADC CONTROL REGISTER 1

bit 10-8 **FORM<2:0>**: Data Output Format bits

011 = Signed Fractional 16-bit (DOUT = 0000 0000 0000 0000 sddd dddd dd00 0000)
010 = Fractional 16-bit (DOUT = 0000 0000 0000 0000 dddd dddd dd00 0000)
001 = Signed Integer 16-bit (DOUT = 0000 0000 0000 0000 ssss sssd dddd dddd)
000 = Integer 16-bit (DOUT = 0000 0000 0000 0000 0000 00dd dddd dddd)
111 = Signed Fractional 32-bit (DOUT = sddd dddd dd00 0000 0000 0000 0000)
110 = Fractional 32-bit (DOUT = dddd dddd dd00 0000 0000 0000 0000 0000)
101 = Signed Integer 32-bit (DOUT = ssss ssss ssss ssss ssss sssd dddd dddd)
100 = Integer 32-bit (DOUT = 0000 0000 0000 0000 0000 00dd dddd dddd)

Figura 53 Formato do dado convertido

Em **"*main.c*"** insira o código para a atividade 1.

```
// INCLUDES
#include "init_sys.h"
#include "my_timer.h"
#include "my_lcd.h"
#include "my_adc.h"

// CONFIGURACAO PARA GRAVACAO
#pragma config FPLLMUL = MUL_16, FPLLIDIV = DIV_2, FPLLODIV = DIV_8, FWDTEN = OFF
#pragma config POSCMOD = HS, FNOSC = PRI, FPBDIV = DIV_2

/// DEFINES
// BOTOES (0:PRESSIONADO 1:LIBERADO)
#define Botao_1 PORTDbits.RD6 //BOTÃO 1
#define Botao_2 PORTDbits.RD7 //BOTÃO 2
#define Botao_3 PORTAbits.RA7 //BOTÃO 3
#define Botao_4 PORTDbits.RD13 //BOTÃO 4
// LED (0:APAGADO 1:ACESO)
#define Led1 LATAbits.LATA0 //LED1
#define Led2 LATAbits.LATA1 //LED2
#define Led3 LATAbits.LATA2 //LED3
#define Led4 LATAbits.LATA3 //LED4
#define Led5 LATAbits.LATA4 //LED5
#define Led6 LATAbits.LATA5 //LED6
#define Led7 LATAbits.LATA6 //LED7
//#define Led8 LATAbits.LATA7 //LED8 // Utilizar como Botao_3 e nao Led_8


int main(void)
{
    char buffer_lcd[17];
    float resultado;

    init_sys(); // Inicializa o sistema

    inicializa_LCD(); // INICIALIZA LCD
    limpar_LCD(); // LIMPA O LCD
    comando_LCD(0x80);
    escreve_frase_LCD("AULA5-MULTIMETRO");

    inicializa_adc(); // Inicializa AD

    while(1){
        while (!AD1CON1bits.DONE); //Aguarda conversao
        resultado = (3.3*ADC1BUF0)/1023; // Faz a conversao do resultado AD para tensao
        sprintf(buffer_lcd,"CH5 : %3.2f V",resultado); // Carrega resultado no buffer
        comando_LCD(0xC0);
        escreve_frase_LCD((char *)buffer_lcd);
    }

    return 0;
}
```

Teste o programa com o MPLAB SIM®.

Após testar o programa conecte o kit EXPLORER16 BR na placa ICD2 BR através do cabo RJ12.

Conecte a placa ICD2BR ao computador através da porta USB.

Alimente o kit EXPLORER16BR com a fonte de alimentação.

Compile o programa (**Ctrl+F10**).

Grave o programa (***Program Target Device***).

Teste o programa.

Perceba que como a amostragem automática está habilitada (AD1CON1bits.ASAM = 1) no final de cada conversão (AD1CON1bits.DONE) o modulo AD inicia automaticamente uma nova amostragem para o próximo *loop*.

Atividade 2

Altere o código da atividade 1 de forma que a conversão automática seja desabilitada. Faça com que a conversão somente seja efetuada se o botão 1 for pressionado. Ou seja, mesmo variando-se o *trimpot* o valor apresentado no *lcd* permanece inalterado até que o botão 1 seja pressionado.

Atividade 3

A placa EXPLORER16BR apresenta além do *trimpot* conectado ao pino RB5/AN5 um sensor de temperatura conectado a entrada analógica conectado ao pino RB4/AN4 (Figura 54). O aluno poderá verificar seu funcionamento baixando o *datasheet* do MCP9700 para estudar a faixa de operação e utilizando o resultado do canal 4 assim como foi feito para o canal 5 nas atividades anteriores.

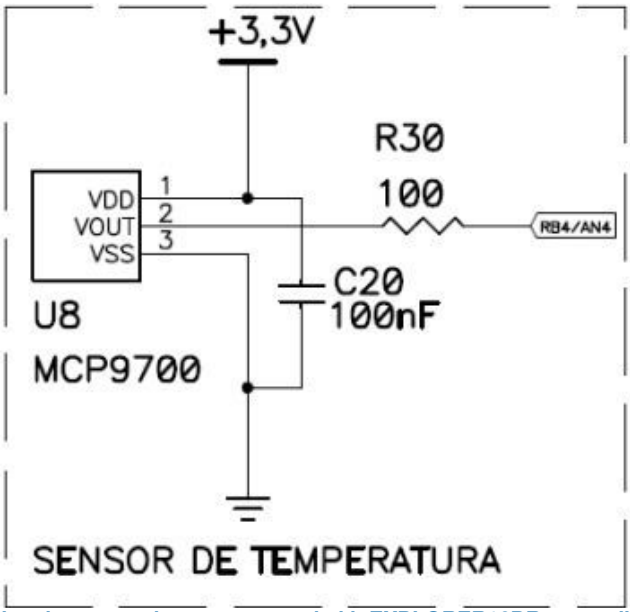

Figura 54 Esquema elétrico do sensor de temperatura do kit EXPLORER16BR para utilizacão do conversor AD

7.6. Aula 6 – Configuração de Interface para Comunicação serial UART com RS-232

Objetivo

Aprender a configurar e programar a interface UART para realização de comunicação serial.

Referências

- *Datasheet* PIC32MX3XX-4XX disponível no CD1 em **"Datasheets >> PIC32"**. Capítulo 19.0: *Universal Asynchronous Receiver Transmitter* (UART)

Introdução

A UART (*Universal Asynchronous Receiver Transmitter*) é um módulo para comunicação serial full-duplex via protocolos do tipo RS232, RS422, RS423, RS485. Nesta prática iremos utilizar o RS232.

A comunicação duplex, ou bidirecional, ocorre quando se têm dois nós (pontos) de comunicação, no caso o PIC e um microcomputador (ex.: PC). Caso a transmissão de dados seja feita somente em um sentido diz-se que a comunicação é simplex (um sistema transmite e o outro recebe). Caso a transmissão seja feita em ambos os sentidos a comunicação é denominada half-duplex (enquanto um dispositivo transmite o outro recebe e vice-versa) ou full-duplex (ambos dispositivos enviam e recebem dados simultaneamente).

Para a comunicação full-duplex o TX (transmissor) do PIC deve estar ligado ao RX (receptor) do PC e o RX do PIC deve estar ligado ao TX do PC. (Figura 55)

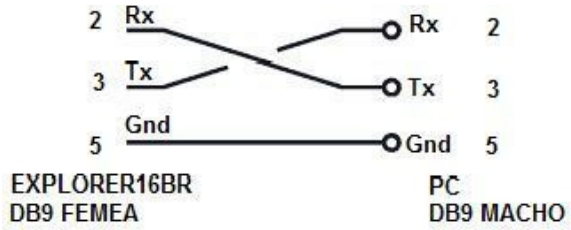

Figura 55 Conexão cabo DB9

Existem alguns cabos do tipo DB9/USB que possuem um *driver* para emular uma porta serial virtual através da conexão USB. Assim poderia-se utilizar um cabo DB9Macho/USB (lado direito da figura) ao invés de um cabo DB9Macho/DB9Femea (lado esquerdo). (Figura 56)

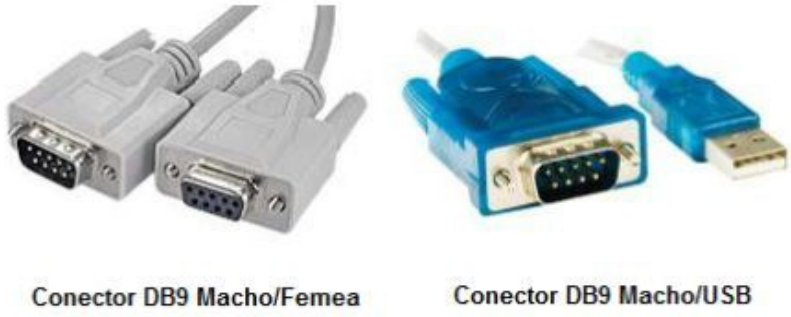

Figura 56 Conector DB9Macho/DB9Femea (esquerda), DB9MAcho/USB (direita).

O RS232 é um protocolo de comunicação serial estabelecido pela EIA (*Electronic Industries Association*) que especifica a temporização dos sinais, tipos de sinais, tensões, conexões e conectores, para a comunicação serial.

As características principais do protocolo RS232 são:

a) **Tensão de Operação**: No protocolo RS232 o nível lógico '0' (zero) vai de -12V a -3V e o nível lógico '1' (um) vai de +3V a +12V. Tensões entre -3V e +3V são indeterminadas. Como usamos um MCU cujo nível de operação vai de 0 V a 3.3 V um *driver* para adequar os níveis de tensão se faz necessário. Na placa EXPLORER16 BR quem faz essa conversão é o CI MAX232. (Figura 57)

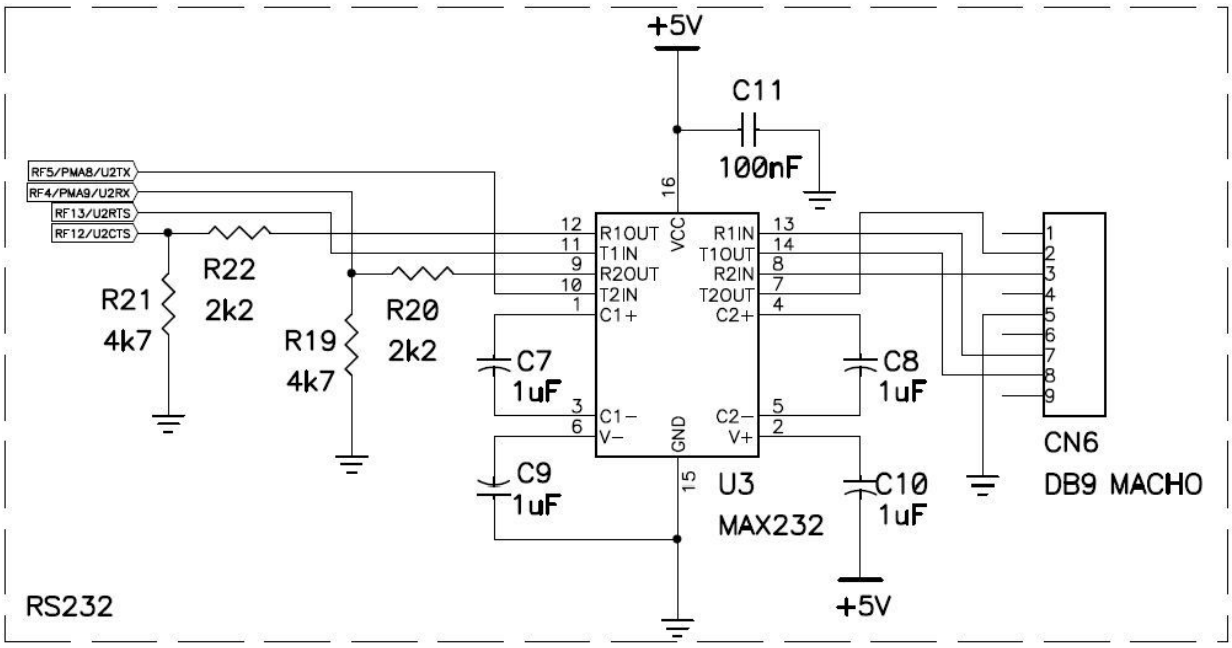

Figura 57 Esquema elétrico do conector DB9 e Driver MAX232 do kit EXPLORER16BR para comunicação RS232.

b) ***Baud Rate***: ou taxa de transferência refere-se a quantidade de bits que serão transmitidos por unidade de tempo. Um *baud rate* tipico de 9600 indica a transmissão de 9600 bits por segundo.

c) **Comunicação Síncrona x Assíncrona**: Na transmissão síncrona um sinal de *clock* é gerado pelo dispositivo mestre a fim de se criar uma temporização regular para o tráfego dos dados. Na comunicação assíncrona (ex: UART) não existe um sinal de *clock* indicando o tempo de envio e recebimento dos dados. O controle de informação nesse caso se dá através de dois pinos a saber: RTS (*ready to send*) e CTS (*clear to send*). O transmissor sinaliza através do pino RTS que deseja enviar informação. O receptor ao perceber a sinalização prepara-se para receber o dado ativando o pino CTS. Quando o transmissor recebe a autorização através do sinal CTS ele inicia a transmissão.

A utilização dos sinais RTS e CTS (pinos RF13/U2RTS e RF12/U2CTS do PIC32MX360F512, respectivamente) requer mais uma conexão no cabo DB9 (pinos 7 e 8, respectivamente), observe a Figura 57.

d) **Tamanho do pacote**: O tamanho do pacote de dados pode variar sendo que sua estrutura é constituída por: 1 *start bit*; 8 ou 9 *bits* de dados; 0 ou 1 *bit* de paridade; 1 ou 2 *bits* de parada.

Tipo	Descrição	Tamanho (bits)
Start Bit	O pacote sempre inicia com o nível lógico zero para indicar o inicio da transmissão.	1
Dado	Os bits de dados (informação) são enviados logo após o start bit.	8 ou 9
Paridade	O *bit* de paridade serve para checar se a informação recebida está correta. Haja visto que a transmissão via cabos[12] pode apresentar distúrbios (ruídos) e modificar o nível lógico de algum *bit* de dado. Ele é adicionado após a transmissão do pacote de dados para verificação de erros.	0 ou 1
Stop Bit	Sinaliza o fim da transmissão.	1 ou 2

Perceba que os periféricos conhecem a duração de cada *bit* devido a configuração do *baud rate*. Por exemplo, seja um *baud rate* de 9600, então um *bit* terá duração de 104µS. Assim que o transmissor sinaliza o *bit* RTS e recebe de volta a confirmação do *bit* CTS enviada pelo receptor a temporização de inicia. Os primeiros 140 µS determinam a duração do *start bit*, os 104 µS seguintes determinam o primeiro *bit* de dado, após mais 104 µS o segundo *bit* de dado é transmitido e assim ocorre até que o *stop bit* seja transmitido.

[12] O protocolo RS232 limita o tamanho máximo do cabo em 15 metros.

Registradores

Os registradores necessários para configurar a UART no MCU PIC32MX360F512L são:

UxBRG (*UART Baud Rate Generator*) - Registrador de configuração da velocidade de transmissão da UARTx. No PIC32MX360F512L existem dois canais de comunicação UART1 e UART2. O valor do registrador UxBRG depende da freqüência de oscilação externa do MCU (Fosc), prescaler do PIC (FPBDiv), o *baud-rate* desejado (BaudRate) e do valor do bit BRGH do registrador U2MODE de acordo com a equação a seguir:

$$UxBRG = \frac{Fosc}{FPBDIV} * (16 * BaudeRate) - 1, se\ o\ bit\ BRGH = 0$$

$$UxBRG = \frac{Fosc}{FPBDIV} * (4 * BaudeRate) - 1, se\ o\ bit\ BRGH = 1$$

Ex.: Para um PIC de 8MHz, prescaler de 2, se BRGH=0 e desejarmos um *Baud rate* de 9600bps temos que configurar: U2BRG = 25; // BRGH=0 → U2BRG = (8000000/2)/(16*9600)-1

UxMODE: Esse registrador e responsável por:

Habilitar (nível logico "1") ou desabilitar (nivel lógico "0") a UART através do bit UxMODEbits.UARTEN (UART ENable). O bit UxMODEbits.UEN serve para habilitar/desabilitar a transmissão/recepção da UART separadamente.

Ex.:

U2MODEbits.UARTEN = 1; // Habilita UART2

U2MODEbits.UEN = 0; // TX e RX habilitados, CTS (*UART Clear to Send*) e RTS (*Request to Send*) controlados via hardware (MAX232)

Habilitar (nível lógico "1") ou desabilitar (nivel lógico "0") o *Auto Baude Rate* através do bit U2MODEbits.ABAUD, (Auto-Baud Enable Bit).

Configurar o modo de velocidade do *baud rate* através do bit U2MODEbits.BRGH. Nível lógico "1" define o modo em alta velocidade e nível lógico "0" define o modo de velocidade padrão. Esse bit e utilizado pelo registrador UxBRG para configuração do *baud rate*.

Configurar o modo de transmissão de dados na UART através dos bits PDSEL (*Parity and Data Selection Bits*) e STSEL (*Stop Selection Bit*).

Ex.: U2MODEbits.PDSEL = 0; // 8 bits de dados, sem paridade

U2MODEbits.STSEL = 0; // 1 stop bit

UxSTA: Esse registrador e responsável por:

Configurar, através dos bits UxSTATbits.UTXISEL (*Tx Interrupt Mode Selection bit*) e URXISEL (*Rx Interrupt Mode Selection Bit*) os momentos em que as interrupções de Tx e Rx devem ser geradas.

Ex.: U2STAbits.UTXISEL = 0; // gera interrupcao a cada Tx.

U2STAbits.URXISEL = 1; // gera interrupcao a cada Rx

Configurar, através dos bits UxSTAbits.URXEN (*Receiver ENable bit*) e UxSTAbits.UTXEN (*Transmit ENable bit*), se os pinos de Rx e Tx serão gerenciados pelo modulo UART. Nivel lógico "1" implica no controle dos pinos pela UART, nível lógico "0" implica no controle dos pinos através da porta.

Atividade 1

Esta atividade tem por objetivo fazer a comunicação entre o computador e o PIC32MX360F512 através do Hyperterminal[13]. Se você for usuário do *Windows Vista* ou *Windows 7* é necessário baixar o software em http://www.hilgraeve.com/. Se você utiliza o *Windows XP* ou anterior ele já está instalado em sua máquina.

Após criar um novo projeto inclua os arquivos ***"init_sys.c"***, ***"init_sys.h"***, ***"my_lcd.c"***, ***"my_lcd.h"***, ***"my_timer.c"*** e ***"my_timer.h"*** utilizados nas praticas anteriores. Agora crie e inclua os arquivos ***"my_rs232.c"*** e ***"my_rs232.h"***. no seu projeto.

No arquivo ***"my_rs232.h"*** insira o seguinte código:

```
#ifndef MY_RS232_H
#define MY_RS232_H

#include <p32xxxx.h> //include para o PIC32MX360F512


void inicializa_UART2(void); //INICIALIZA UART2
unsigned int RxUART2(void); //LE UART
void TxUART2(unsigned int data); //ESCREVE UART

#endif
```

No arquivo ***"my_rs232.c"*** insira o seguinte código:

```
#include "my_rs232.h"

void inicializa_UART2(void)
{

   // Se BRGH=0 -> U2BRG = (Fosc/FPBDIV)/(16*BaudRate) - 1
   // Se BRGH=1 -> U2BRG = (Fosc/FPBDIV)/(4*BaudRate) – 1
   U2BRG = 25; // BRGH=0 -> U2BRG = (8000000/2)/(16*9600)-1
   U2MODEbits.UARTEN = 1; // Habilita UART2
   U2MODEbits.UEN = 0; // TX e RX habilitados, CTS e RTS controlados via hardware (MAX232)
   U2MODEbits.ABAUD = 0; // Desabilita o autobaudrate
   U2MODEbits.BRGH = 0; // Configuração do BRGH
   U2MODEbits.PDSEL = 0; // 8 bits de dados, sem paridade
   U2MODEbits.STSEL = 0; // 1 stop bit
   U2STAbits.UTXISEL0 = 0; // gera interrupcao a cada Tx
   U2STAbits.UTXISEL1 = 0;

   U2STAbits.URXISEL = 1; // gera interrupcao a cada RX

   U2STAbits.UTXEN = 1; // Habilita pino TX
   U2STAbits.URXEN = 1; // Habilita pino RX
}


unsigned int RxUART2(void) // Lê UART
{
   if(U2MODEbits.PDSEL == 3) return (U2RXREG);
   else return (U2RXREG & 0xFF);
}


void TxUART2(unsigned int data) // Escreve UART
{
   if(U2MODEbits.PDSEL == 3) U2TXREG = data;
   else U2TXREG = data & 0xFF;
}
```

[13] Hyperterminal (Hilgrave 2010) é um programa capaz de conectar sistemas através de redes TCP/IP, DIAL-UP Modems e portas de comunicação COM.

A configuração do registrador para definição do *Baud Rate* pode ser melhor interpretada a partir do exemplo extraído do *datasheet*. (Figura 58)

EXAMPLE 19-1: BAUD RATE ERROR CALCULATION (BRGH = 0)

```
Desired Baud Rate      =   Fpb/(16 (UxBRG + 1))
Solving for UxBRG value:
    UxBRG              =   ( (Fpb/Desired Baud Rate)/16) - 1
    UxBRG              =   ((4000000/9600)/16) - 1
    UxBRG              =   [25.042] = 25
Calculated Baud Rate   =   4000000/(16 (25 + 1))
                       =   9615
Error                  =   (Calculated Baud Rate - Desired Baud Rate)
           Desired Baud Rate
                       =   (9615 - 9600)/9600
                       =   0.16%
```

Figura 58 Exemplo de cálculo do Baud Rate

Em "*main.c*" insira o código para a atividade 1.

```
// INCLUDES
#include "init_sys.h"
#include "my_timer.h"
#include "my_lcd.h"
#include "my_rs232.h"

// CONFIGURACAO PARA GRAVACAO
#pragma config FPLLMUL = MUL_16, FPLLIDIV = DIV_2, FPLLODIV = DIV_8, FWDTEN = OFF
#pragma config POSCMOD = HS, FNOSC = PRI, FPBDIV = DIV_2

/// DEFINES
// BOTOES (0:PRESSIONADO 1:LIBERADO)
#define Botao_1 PORTDbits.RD6 //BOTÃO 1
#define Botao_2 PORTDbits.RD7 //BOTÃO 2
#define Botao_3 PORTAbits.RA7 //BOTÃO 3
#define Botao_4 PORTDbits.RD13 //BOTÃO 4
// LED (0:APAGADO 1:ACESO)
#define Led1 LATAbits.LATA0 //LED1
#define Led2 LATAbits.LATA1 //LED2
#define Led3 LATAbits.LATA2 //LED3
#define Led4 LATAbits.LATA3 //LED4
#define Led5 LATAbits.LATA4 //LED5
#define Led6 LATAbits.LATA5 //LED6
#define Led7 LATAbits.LATA6 //LED7
//#define Led8 LATAbits.LATA7 //LED8 // Utilizar como Botao_3 e nao Led_8

int main(void)
{
   char caractere;

   init_sys(); // Inicializa o sistema

   inicializa_LCD(); // INICIALIZA LCD
   limpar_LCD(); // LIMPA O LCD

   inicializa_UART2();
   comando_LCD(0x80);
   escreve_frase_LCD("COMUNICACAORS232");
   comando_LCD(0xC0);
   escreve_frase_LCD("Caractere:");

   putsUART2("UNIVERSIDADE FEDERAL DE MINAS GERAIS\r\n");
   putsUART2("AULA 6 - COMUNICACAO SERIAL VIA RS232\r\n");
   putsUART2("Digite algo no teclado para ver o caractere escrito na tela e no lcd\r\n");

   while(1) {

      if(U2STAbits.URXDA){ // Espera dado

      caractere = RxUART2(); // Recebe dado do pc
      TxUART2(caractere); // Envia dado para pc
```

```
    comando_LCD(0xCD);
    dado_LCD(caractere); // Escreve dado no LCD

    }

  }
}
```

Conecte o kit EXPLORER16 BR na placa ICD2 BR através do cabo RJ12.
Conecte a placa ICD2 BR ao computador através da porta USB.
Alimente o kit EXPLORER16 BR com a fonte de alimentação.
Compile o programa (Ctrl+F10).
Grave o programa (*Program Target Device*).
Para testar o programa desconecte a placa ICD2 BR. Em seguida conecte o cabo DB9 Macho na placa EXPLORER16 BR e a outra extremidade (DB9 ou USB) no computador.
Abra o Hyperterminal. (Figura 59)

Figura 59 Hyperterminal

Dê um nome para sua conexão e clique ***OK***.
Na tela que se segue escolha a porta serial (COMx) em que a placa EXPLORER16 BR está conectada. (Figura 60) Para cada computador a porta em questão poderá variar. Quando se usa um conector USB para emular a porta serial poderá aparecer diferentes numerações das portas como COM4, COM7, COM8, etc.

Figura 60 Hyperterminal

Selecione os parâmetros de comunicação da mesma forma que foram programados no PIC. (Figura 61)

Figura 61 Parâmetros da comunicação RS232.

Clique ***OK***. Pronto! A comunicação está estabelecida. Para finalizá-la clique no ícone "Desconectar" (Figura 62)

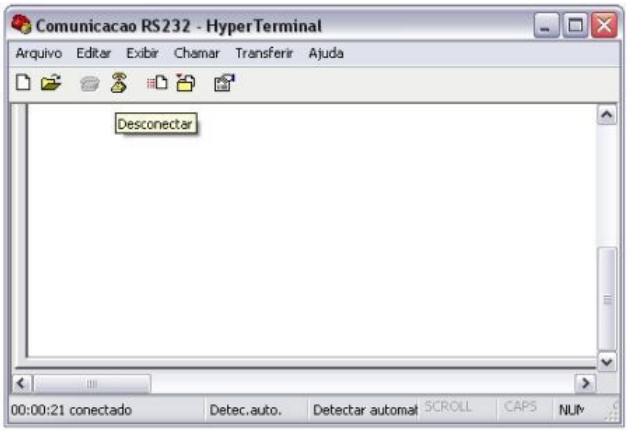

Figura 62 Finalizar comunicação RS232.

Para testar seu programa e ver a mensagem inicial na tela do *Hyperterminal* reste a placa EXPLORER16 BR. Assim que você resetar seu programa e escrever algo as seguintes mensagens devem aparecer no *Hyperterminal* e no LCD. (Figura 63)

Figura 63 Comunicação RS232. Aula 6 - Atividade 1.

Atividade 2

Modifique o *bit* ***U2MODEbits.BRGH*** de '0' (zero) para '1' (um) e, modificando o valor do registrador ***U2BRG***, altere a velocidade de transmissão de 9600 bps para 19200 bps. Dica: A fórmula do cálculo para U2BRG, considerando-se U2MODEbits.BRGH=1, pode ser obtida no *datasheet*.

Além de modificar a taxa de transmissão altere a transmissão para 8 bits de dados com bit de paridade par.

Atividade 3

Faça um programa de controle por acesso a senha composta por 3 dígitos.
A senha deverá ser armazenada em uma variável interna ao PIC32MX360F512.
A cada caractere digitado um asterisco deverá aparecer na tela do LCD.
O usuário terá 3 chances para inserir a senha correta que esta gravada no programa. Caso ele acerte a mensagem ***'SENHA CORRETA'*** deverá aparecer na tela do computador. Para cada tentativa mal sucedida a mensagem ***'SENHA INCORRETA – Tentativa x de 3'*** deverá aparecer. Caso a senha seja inserida incorretamente 3 vezes um alarme deve ser acionado (obs: o alarme será a mensagem ***'ALARME!!!'*** na tela do computador).
As Figura 64, 65, 66 e Figura 677 ilustram o funcionamento do programa.

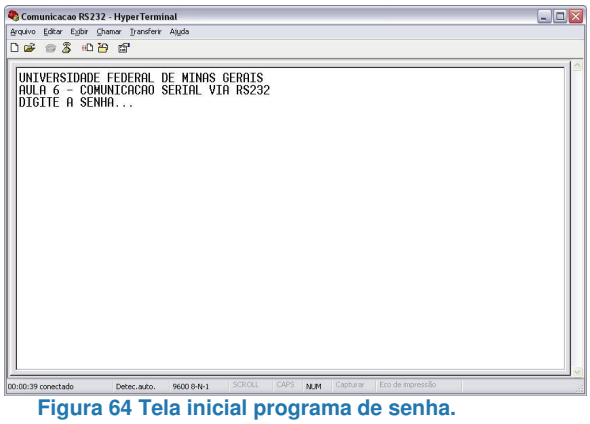

Figura 64 Tela inicial programa de senha.

Figura 65 Tela LCD programa de senha.

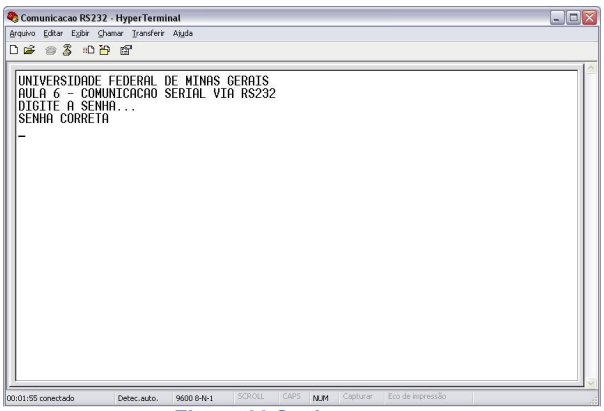

Figura 66 Senha correta.

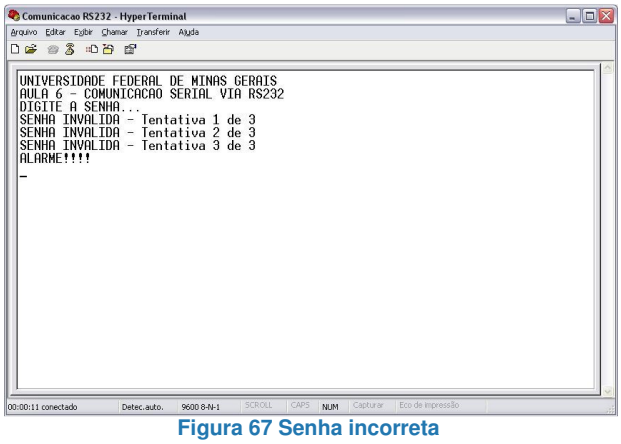

Figura 67 Senha incorreta

7.7. Aula 7 – Configuração do Periférico de Modulação de Largura de Pulsos (PWM).

Objetivo

Aprender a configurar e programar o periférico PWM.

Referências

- *Datasheet* PIC32MX3XX-4XX disponível no CD1 em "Datasheets >> PIC32". Capítulo 16.0: *Output Compare*.
- Aula 3.

Introdução

A modulação PWM possui várias aplicações dentre as quais podemos citar os conversores estáticos de potência nos quais a partir da abertura e fechamento de chaves pode-se controlar o fluxo de potência entre da fonte à carga. Outra aplicação também bastante comum é o controle de motores de passo utilizados em impressoras como exemplo.

Para gerar um sinal PWM precisamos do sinal modulante e do sinal da portadora. A portadora é uma onda do tipo dente de serra com amplitude e freqüência bem definidas. A onda modulante é um sinal constante cuja amplitude vai deste o valor mínimo da portadora até seu valor máximo.

O sinal PWM é gerado fazendo-se a comparação da onda modulante com a portadora. Caso a onda modulante seja maior que a portadora a saída vai para nível lógico baixo, caso a modulante seja menor que a portadora a saída vai para nível lógico alto.

Como exemplo considere uma portadora com valor mínimo '0' (zero), valor máximo '1' (um) e cuja freqüência seja 2 KHz. Considere ainda uma modulante cuja amplitude seja '0.7'. Observe na Figura 68 que nos instantes de tempo em que a modulante é superior a dente de serra a saída está em nível lógico baixo e quando a modulante é menor a saída está em nível lógico alto.

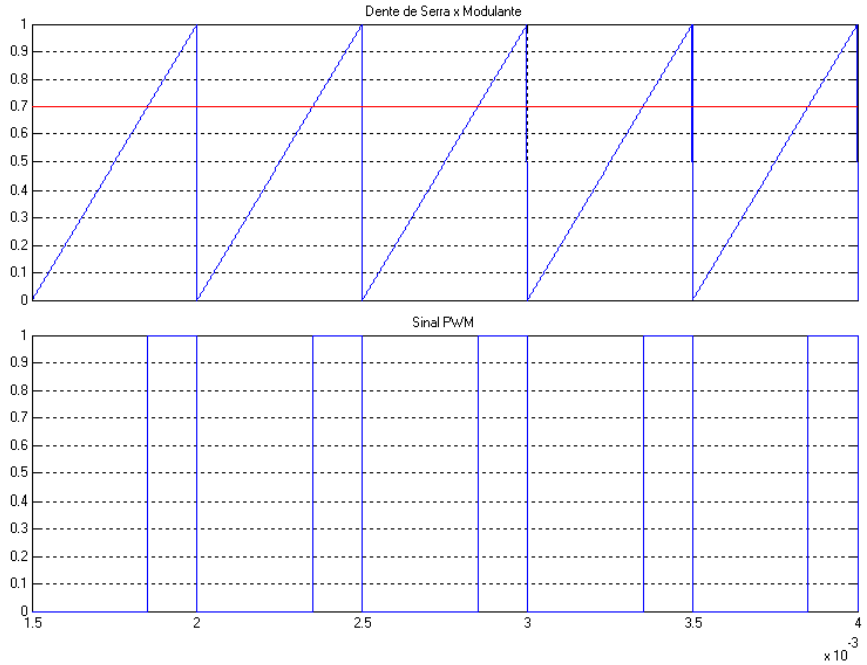

Figura 68 Geração do sinal PWM

Para caracterizar o sinal PWM precisamos de duas grandezas: a freqüência de operação, dada pela freqüência da dente se serra e o ciclo de trabalho (*duty cycle*).

O ciclo de trabalho é a percentagem do tempo em que o sinal PWM permanece em estado ativo. No exemplo apresentado anteriormente a saída permaneceu em nível lógico alto durante $1{,}5*10^{-4}$ s o que representa 30% dos $5*10^{-4}$ s do período da dente se serra. Assim o ciclo de trabalho (ou *duty cycle*) é representado por:

$$\boldsymbol{D = \frac{T_{on}}{T}}$$

Onde T_{ON} é o tempo em que o sinal PWM permanece em nível lógico alto é T é o período da portadora.

O módulo *output compare* disponível no periférico PWM do PIC nos permite gerar até cinco sinais PWM diferentes, informando apenas a freqüência de operação e o ciclo de trabalho.

O modo de funcionamento é basicamente o seguinte: O usuário configura o *timer* com um valor desejado para o período no registrador PRx. Este mesmo valor é passado ao registrador OCxRS do módulo *output compare* e representa o período da onda portadora (dente de serra). Em seguida o valor do sinal modulante é inicializado no registrador OCxR. O módulo *output compare* faz a comparação entre o valor programado pelo usuário (OCxR) e o valor do *timer*. Quando o valor do *timer* torna-se maior que o valor armazenado no registrador (OCxR) o modulo *output compare* gera um evento baseado no modo de operação pré-definido e leva a saída do PWM para nível lógico alto. Observe o diagrama em blocos do módulo *output compare* na Figura 69 a seguir.

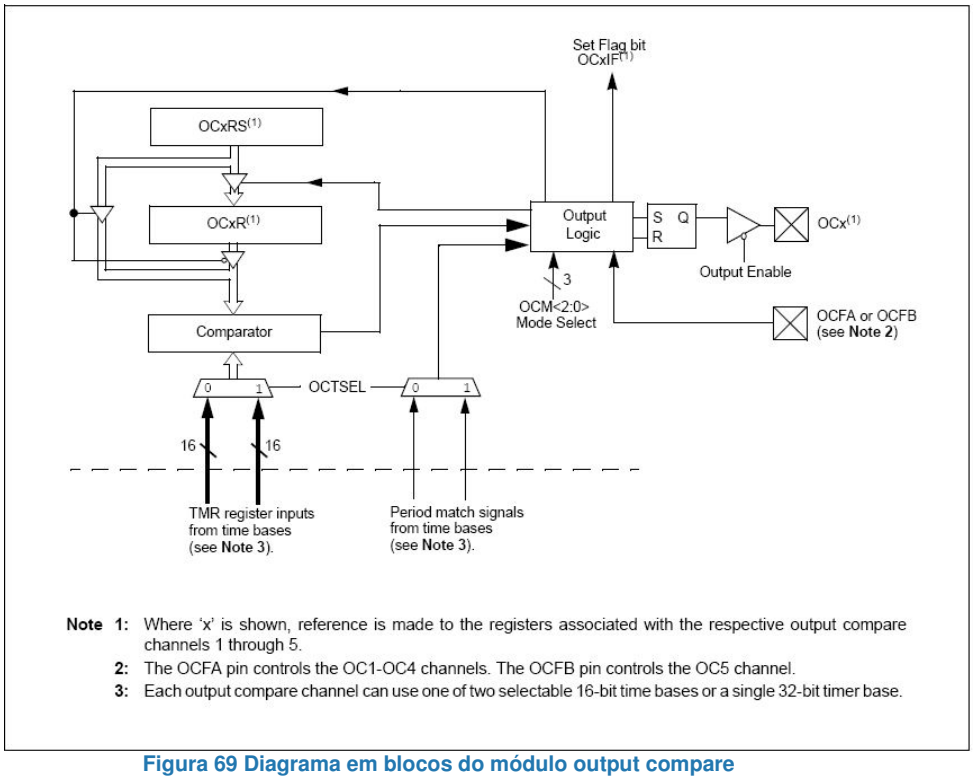

Figura 69 Diagrama em blocos do módulo output compare

Para facilitar a compreensão considere o *timer* como sendo a dente se serra (portadora) e o valor armazenado no registrador do *output compare* como a referência (modulante).

Além dos registradores para configuração do *timer* mostrados na aula prática 3 devemos configurar o módulo *output compar*e através dos registradores apresentados na próxima seção.

Registradores

Os registradores necessários para configurar e gerar o sinal PWM em um dos 5 módulos *Output Compare* presentes no PIC32MX360F512L são:

PRx: Registrador de Período associado ao contador x. (Ver aula 3)

OCxRS (*Output Compare x Secondary Register*) – O valor passado a esse registrador representa o período da portadora do sinal PWM (dente de serra). Esse valor deve ser igual ao valor passado ao registrador PRx.

OCxR (*Output Compare x Primary Register*) – O valor passado a esse registrador (sinal modulante) sera comparado ao valor da portadora, previamente programado pelo usuário através dos registradores PRx e OCxRS.

OCxCON (*Output Compare x Control Register*) - Este registrador é responsável por:

Habilitar (nível lógico "1") ou desabilitar (nivel lógico "0") o módulo *output compare*, através do bit OCxCONbits.ON.

Definir através do bit OxCONbits.OCTSEL se o *timer* 2 (nível lógico "0") e/ou *timer* 3 (nivel lógico "1") serão utilizados para a comparação.

Definir se o pino irá de nível lógico baixo para nível lógico alto ou vice-versa e se a geração de pulsos será contínua ou não, através dos bit OCxCONbits.OCM.

Atividade 1

Após criar um novo projeto inclua os arquivos *"init_sys.c" e "init_sys.h"* utilizados nas praticas anteriores.

Em "*main.c*" insira o código para a atividade 1.

```
1 // INCLUDES
2 #include <p32xxxx.h> //include para o PIC32MX360F512
3 #include <plib.h>
4
5 // CONFIGURACAO PARA GRAVACAO
6 #pragma config FPLLMUL = MUL_16, FPLLIDIV = DIV_2, FPLLODIV = DIV_8, FWDTEN = OFF
7 #pragma config POSCMOD = HS, FNOSC = PRI, FPBDIV = DIV_4 //(PBCLK is SYSCLK divided by 8,4,2,1)
8
9
10
11 void init_TMR2(void)
12 {
13     T2CONbits.TON = 0; // Timer2 desligado
14     TMR2 = 0; // Zera o timer para início de contagem
15
16     // Timer2 Prescaler
17     // TCKPS -> Prescaler
18     // 0 -> 1:1
19     // 1 -> 1:2
20     // 2 -> 1:4
21     // 3 -> 1:8
22     // 4 -> 1:16
23     // 5 -> 1:32
24     // 6 -> 1:64
25     // 7 -> 1:256
26     T2CONbits.TCKPS = 6; // Prescaler 1:256
27
28     // Configura o registrador de período
29     // PR2 = (Fosc * Tempo)/(FPBDIV * PS)
30     // PR2 = (8000000 * 0.1)/(4 * 64) = 3125
31     PR2 = 3125;
32
33     T2CONbits.TCS = 0; // Modo timer (clock interno)
34     T2CONbits.TGATE = 0;
35
36     IFS0bits.T2IF = 0; // Limpa o flag
37
38     T2CONbits.TON = 1; // Timer2 ligado
39 }
40
41
42 int main(void)
43 {
44
45     init_sys();
46
47     // Configura Timer 2 (100ms)
48     init_TMR2();
49
50     // Configura modulo Output Compare
51     OC5CONbits.ON = 1; // Habilita PWM 5
52     OC5CONbits.OC32 = 0; // Timer 16bits
53     OC5CONbits.OCTSEL = 0; // Selecao do Timer2 como referencia
54     OC5CON |= 0x5; // Pulsos Continuos
55     OC5CON &= 0xFFFFFFF5; // Pulsos Continuos
56     OC5RS = 3125; // Periodo = PRx
57     OC5R = 2500; // Duty Cycle 20%
58
59     while(1);
60
61     return (0);
62 }
63
```

O código acima possui uma freqüência de operação de 100 ms devido a configuração do *timer* 2. Perceba, na linha 31, que o período equivale a 3125. Desta forma para obtermos um ciclo

de trabalho de 20% precisamos inicializar o registrador OC5PR, linha 57, com o valor: 0.8*3125 = 2500. Onde 0.8 representa o complemento do ciclo de trabalho desejado (0.2). Caso você altere a configuração do módulo *output compare* para transição de nível lógico alto para baixo, ao invés de baixo para alto, o valor do registrador OC5R deverá ser 0.2*3125=625.

Teste o programa com o MPLAB® SIM.

Após testar o programa conecte o kit EXPLORER16 BR na placa ICD2 BR através do cabo RJ12.

Conecte a placa ICD2BR ao computador através da porta USB.

Alimente o kit EXPLORER16 BR com a fonte de alimentação.

Compile o programa (Ctrl+F10).

Grave o programa (*Program Target Device*).

Atividade 2

Altere o código da atividade anterior para uma freqüência de operação de 0,25 Hz com ciclo de trabalho de 70%. Ateste o funcionamento do módulo PWM com o auxílio do osciloscópio. Para isso configure um pino do PIC para apresentar a saída do módulo PWM.

Atividade 3

Configure o módulo PWM para fazer o controle de velocidade (pelo menos 3 velocidades diferentes) e acionamento de uma ventoinha.

ANEXOS

Anexo I : Adicionando arquivos durante a criação do projeto

No capítulo 5 foi mostrado que é possível adicionar arquivos durante a criação de um projeto. Será mostrado nesta seção as diferentes maneiras de se adicionar o arquivo. Para isso crie uma pasta chamada **"myProject1"** e outra chamada **"myProject2"**.

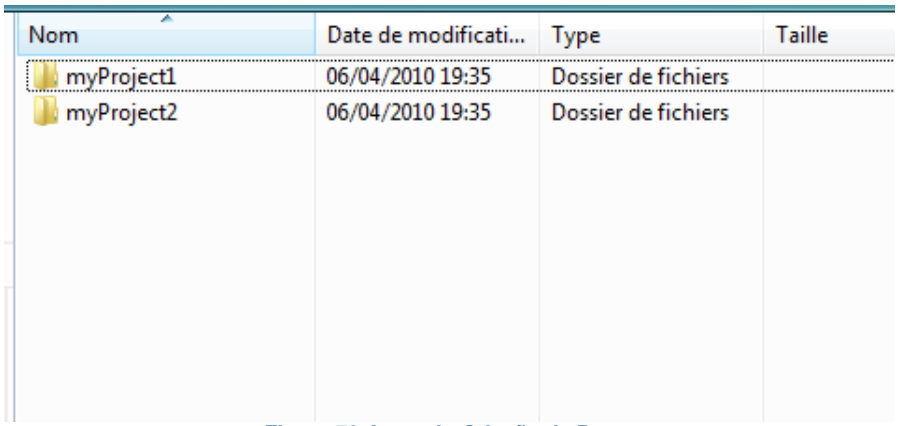

Figura 70 Anexo I - Criação de Pasta

Dentro da pasta ***"myproject1"*** crie 4 arquivos chamados: ***"ArqA.c"***, ***"ArqU.c"***, ***"ArqS.c"*** e ***"ArqC.c"***. Escreva a seguinte mensagem em cada arquivo: ***"Arquivo do tipo 'X' contido na pasta myProject1"***. Onde X deve ser substituído por 'A', 'U', 'S' ou 'C' de acordo com o nome do arquivo. (Figura 71)

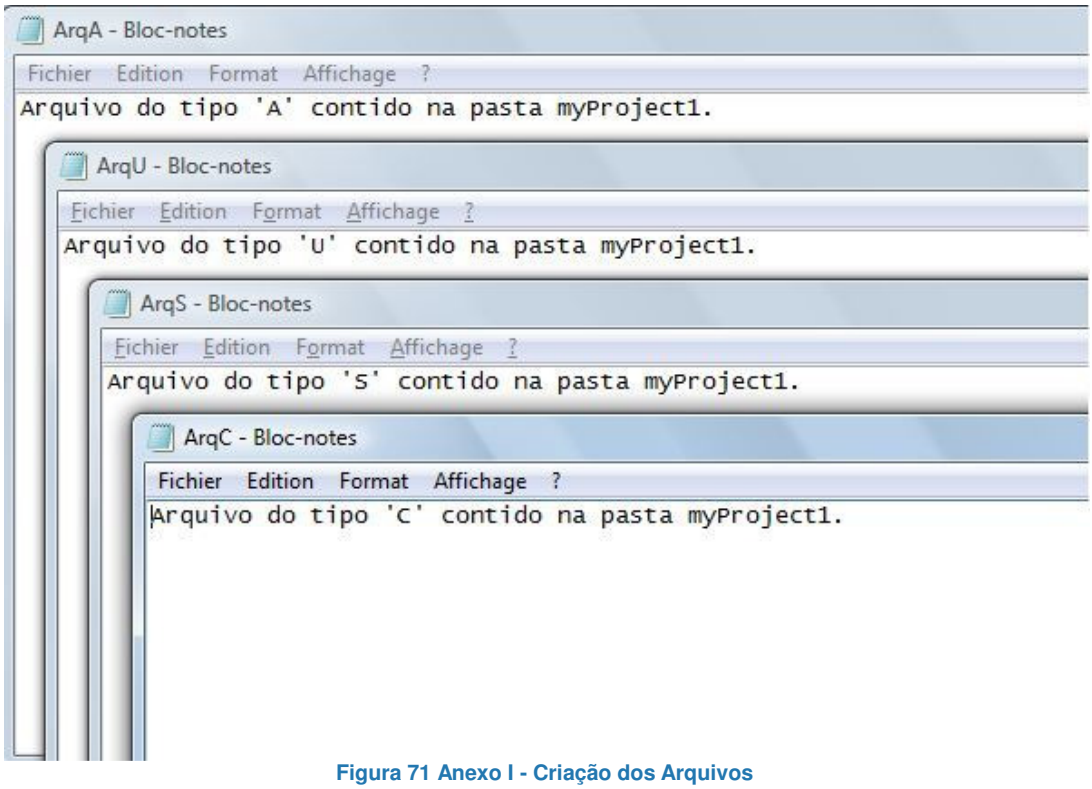

Figura 71 Anexo I - Criação dos Arquivos

Agora abra o MPLAB® e crie um novo projeto de acordo com as diretrizes do capitulo 5 e salve-o na pasta "myProject2". Quando a janela para adição de arquivos aparecer inclua os arquivos ***"ArqA.c"***, ***"ArqU.c"***, ***"ArqS.c"*** e ***"ArqC.c"*** que estão na pasta ***"myProject1"***.

Na coluna da direita tome cuidado para a letra corresponder ao tipo de arquivo. (Figura 72)

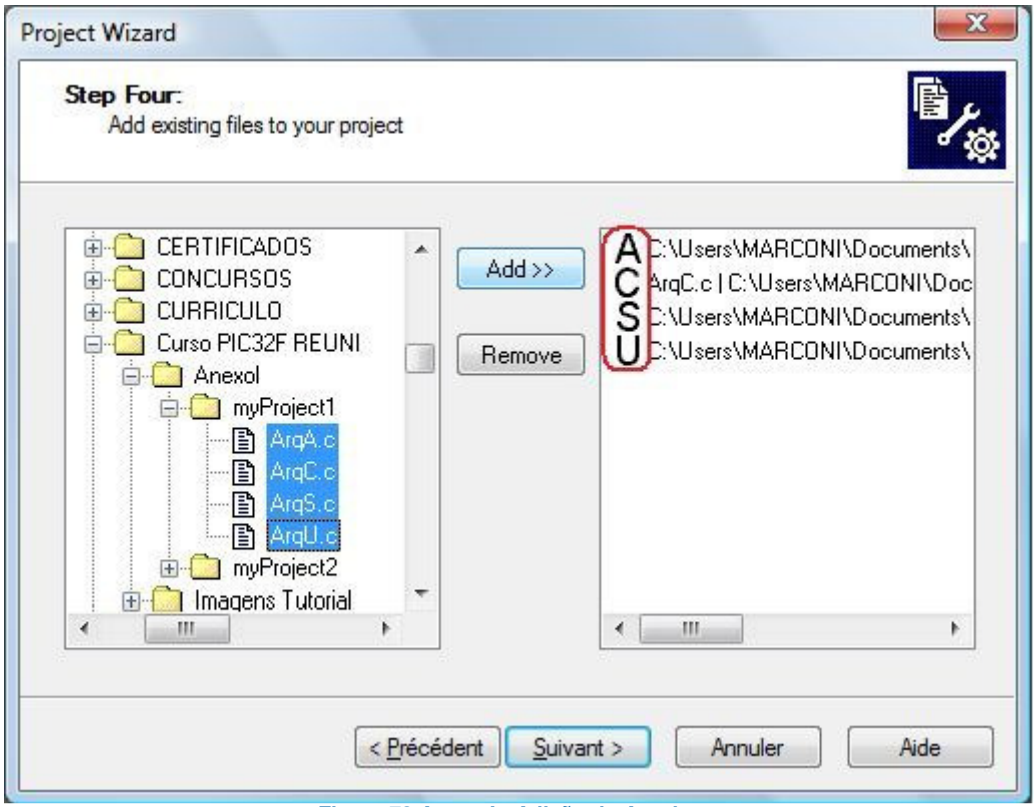

Figura 72 Anexo I - Adição de Arquivos

Finalizada a criação do projeto, a seguinte tela deverá aparecer indicando que os arquivos foram inclusos. (Figura 73)

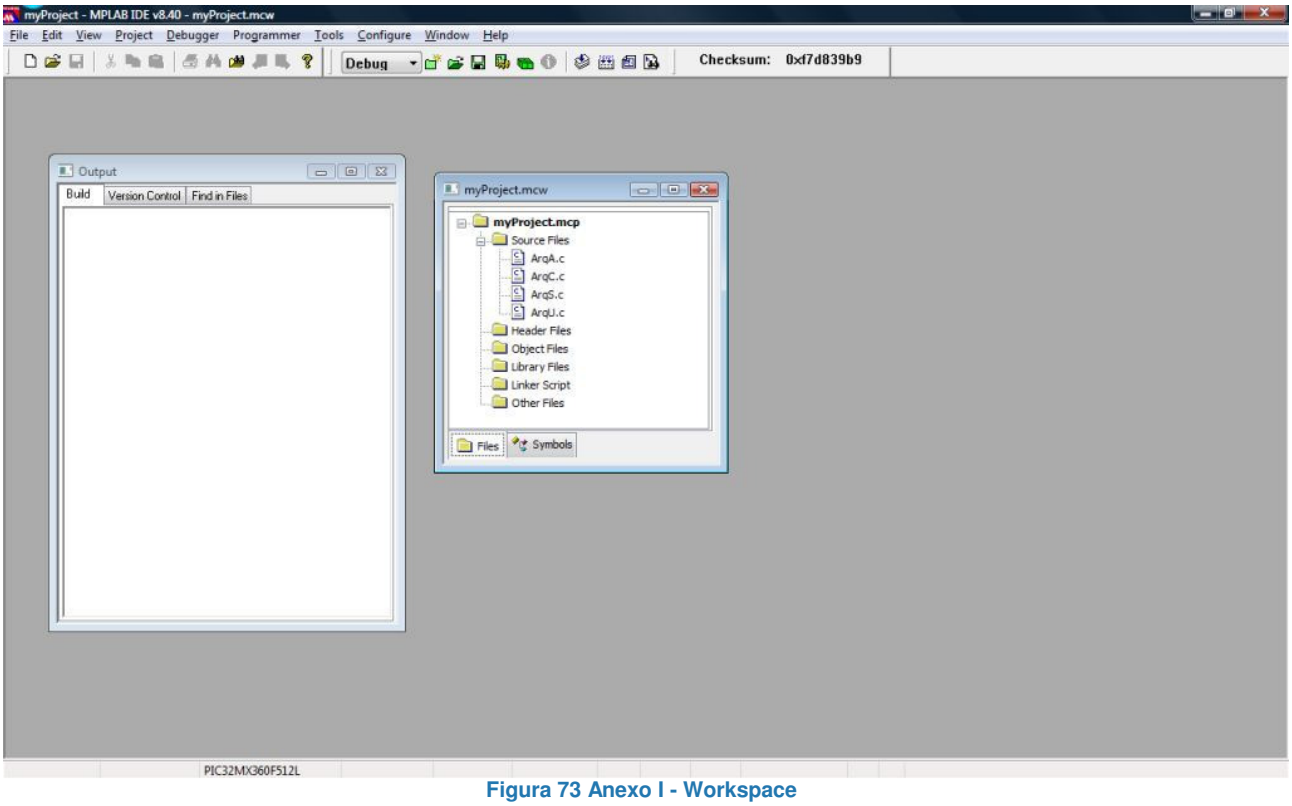

Figura 73 Anexo I - Workspace

Agora clique em cada um dos arquivos e modifique a frase ***"Arquivo do tipo 'X' contido na pasta myProject1"*** para ***"Arquivo do tipo 'X' contido na pasta myProject2"***, onde 'X' deve ser substituído por "A", "U", "S" ou "C" de acordo com o nome do arquivo. (Figura 74)

Salve o projeto (**Ctrl+Shift+S**).

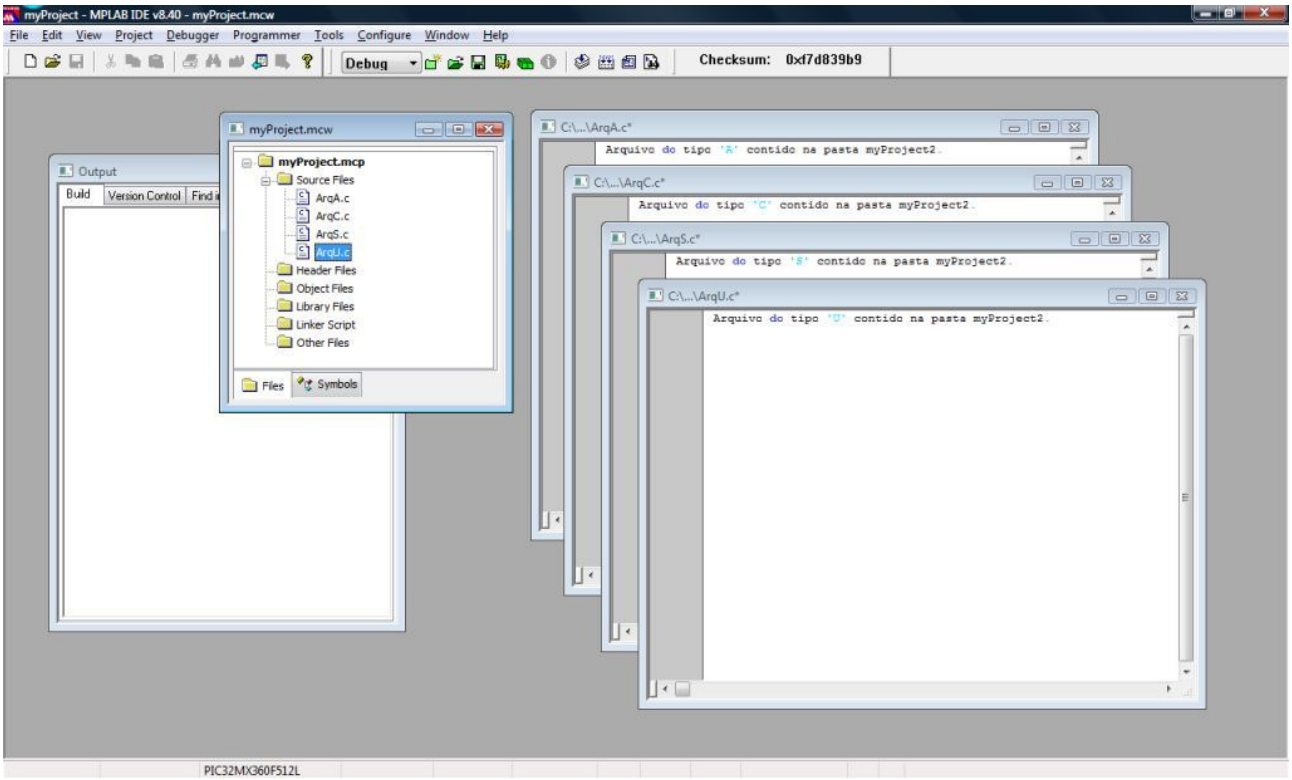

Figura 74 Anexo I - Modificação dos Arquivos

Agora, para ver o que ocorreu, vá na pasta "***myProject1***" e olhe o conteúdo de cada um dos 4 arquivos. (Figura 75)

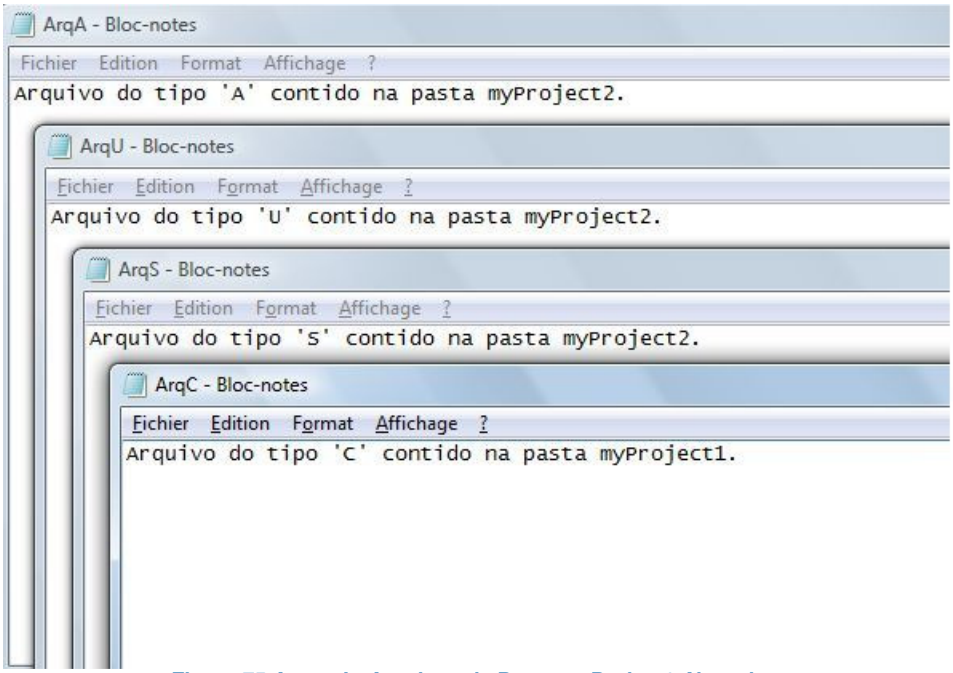

Figura 75 Anexo I - Arquivos da Pasta myProject1 Alterados

Perceba que a alteração da frase **"...myProject1"** por **"...myProject2"**, (Figura 74), alterou também os arquivos **"ArqA.c"**, **"ArqU.c"** e **"ArqS.c"** contidos na pasta **"myProject1"**. (Figura 75) Isso ocorreu porque somente o tipo 'C' faz uma cópia real do arquivo para a pasta **"myProject2"**, os arquivos do tipo 'A', 'U' e 'S', também utilizados no projeto, não foram copiados. O que foi copiado foram somente suas referencias. Desta forma a alteração dos arquivos do tipo 'A', 'U' e 'S' irá modificar os arquivos contidos na pasta **"myProject1"** e não **"myProject2"**.

Para confirmar o que foi dito acima abra a pasta **"myProject2"** e veja que além do projeto criado existe somente o arquivo do tipo 'C'. (Figura 76)

Nom	Date de modificati...	Type	Taille	Mots
ArqC	06/04/2010 20:09	Fichier C	1 Ko	
myProject	06/04/2010 20:01	Microchip MPLAB...	2 Ko	
myProject.mcs	06/04/2010 20:01	Fichier MCS	1 Ko	
myProject	06/04/2010 20:01	Microchip MPLAB...	32 Ko	

Figura 76 Anexo I - Somente o Arquivo do Tipo 'C' é Criado

Desta forma deve-se tomar cuidado ao incluir arquivos durante a criação de projetos. Caso se queira criar somente uma referencia escolha a opção 'A', 'U' ou 'S'. Caso queira criar uma cópia real do arquivo para dentro da pasta do projeto escolha a opção 'C'. (Figura 77)

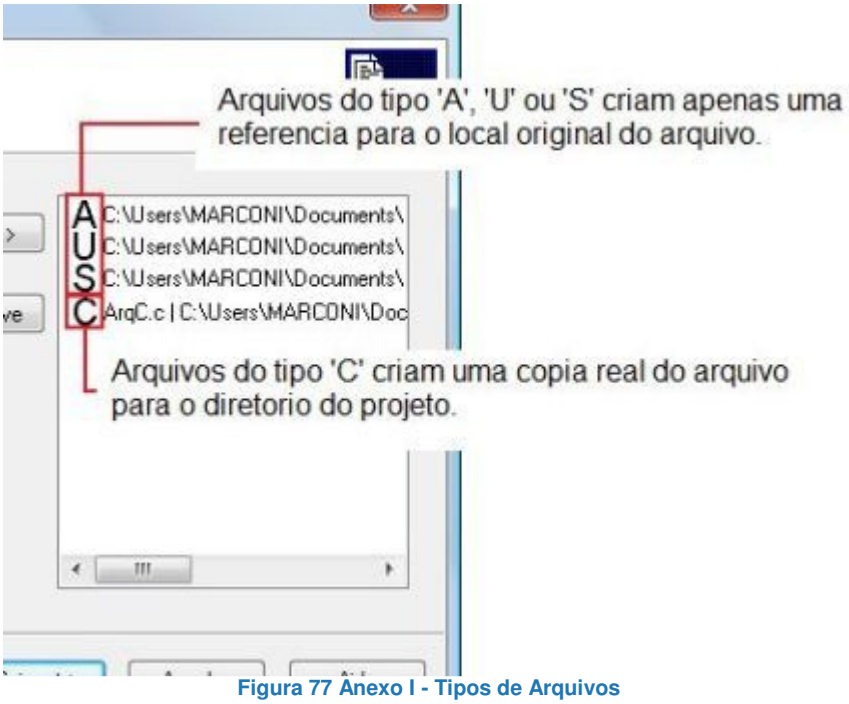

Figura 77 Anexo I - Tipos de Arquivos

Anexo II : Adicionando arquivos depois da criação do projeto

Para melhor compreensão da adição de arquivos leia o Anexo I : Adicionando arquivos durante a criação do projeto.

A adição de arquivos durante a criação do projeto pode ser feita criando-se uma cópia do arquivo para dentro da pasta na qual o projeto se encontra ou apenas uma referência para o arquivo.

Porém, a adição de arquivos ('.c', '.h', '.lib', etc) depois que o projeto foi criado só pode ser feita de uma maneira, que é através da referência do arquivo. Ou seja, se fizermos: **"Project >> Add Files To Project"** somente a referência do arquivo será criada.

Caso desejamos um '.h' que fizemos em um projeto anterior mas precisamos fazer algumas alterações no código recomenda-se fazer uma cópia deste arquivo para dentro da pasta do projeto (**Ctrl+C, Ctrl+V**) e ai sim adicioná-lo ao projeto.

Anexo III : Utilização do Socket para Gravação/Depuração

O Socket deve ser utilizado para PIC do tipo DIP (Dual In Line Package) de 8,14,18,28 e 40 pinos.

Caso o PIC seja de 8,14 ou 18 pinos os dois jumpers 'A' devem ser selecionados. Se o PIC for 28 ou 40 pinos os dois jumpers 'B' devem ser selecionados. (Figura 78)

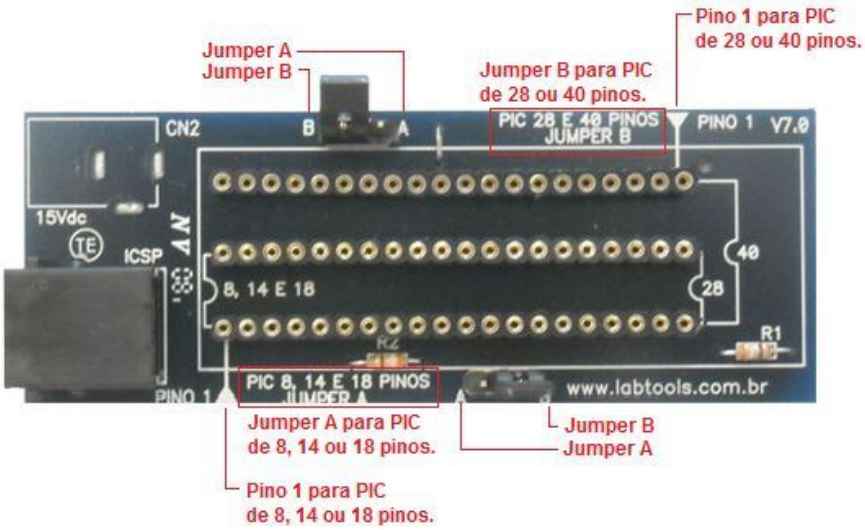

Figura 78 Utilização do Socket para PIC tipo DIP

Depois que o PIC foi inserido no *socket* basta conectá-lo a placa ICD2 BR através do conector RJ12 (Figura 4) e seguir os passos de gravação conforme mostrado no capitulo 6.

Anexo IV : MPLAB® SIM

MPLAB® SIM (MPLAB® *Simulator*) é um dos componentes (*software*) do MPLAB® IDE. Este simulador roda diretamente na plataforma MPLAB® e é utilizado para testar o programa a ser gravado no PIC.

Para utilizá-lo execute os passos a seguir:

Vá em: ***"Debugger >> Select Tool >> 4 MPLAB SIM"***. (Figura 79)

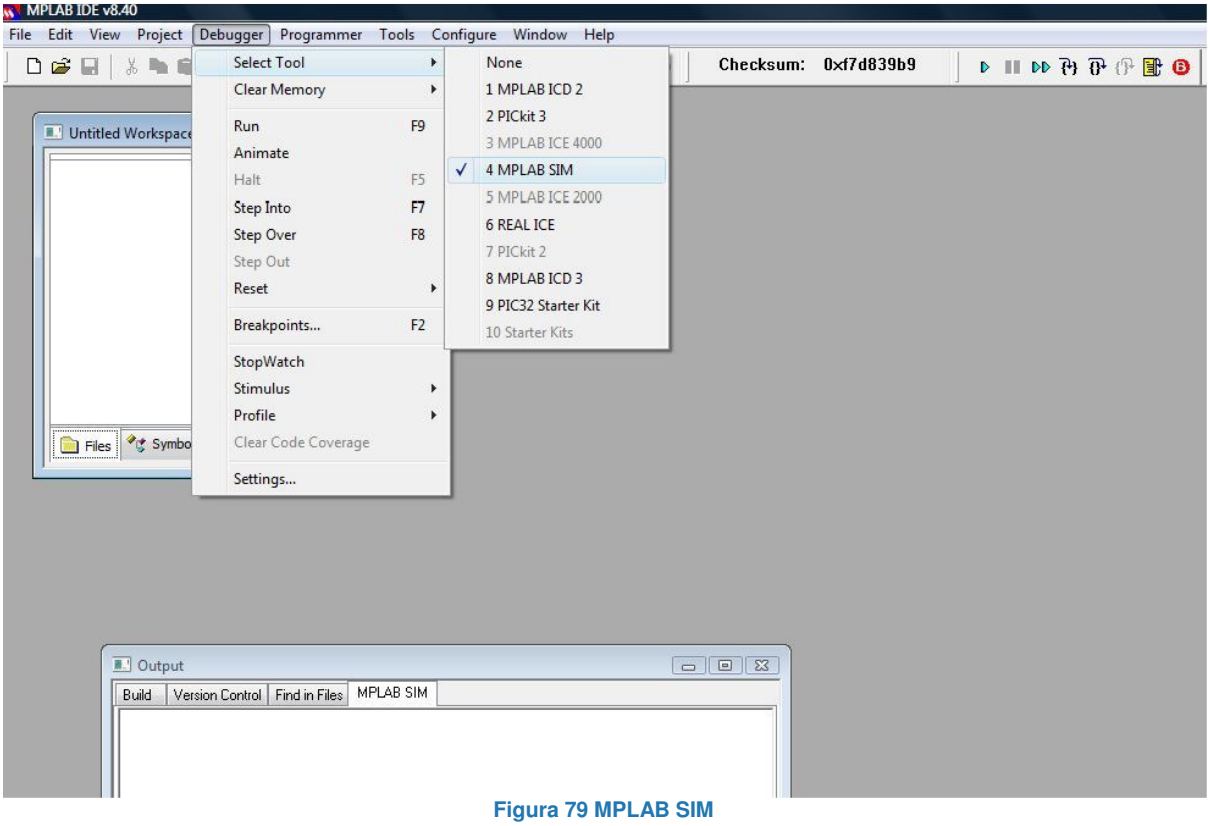

Figura 79 MPLAB SIM

Agora crie um projeto e no arquivo **"main.c"** inclua o seguinte código.[14] Perceba que esse código é o mesmo apresentado na atividade 1 da primeira aula.

```
// INCLUDES
#include <p32xxxx.h> //include para o PIC32MX360F512L

// CONFIGURACAO PARA GRAVACAO
#pragma config FPLLMUL = MUL_16, FPLLIDIV = DIV_2, FPLLODIV = DIV_8, FWDTEN = OFF
#pragma config POSCMOD = HS, FNOSC = PRI, FPBDIV = DIV_2

// 0 Pressionado / 1 Liberado
#define Botao_1 PORTDbits.RD6 //BOTÃO 1
#define Botao_2 PORTDbits.RD7 //BOTÃO 2
#define Botao_3 PORTAbits.RA7 //BOTÃO 3
#define Botao_4 PORTDbits.RD13 //BOTÃO 4

// 0 Apagado / 1 Aceso
#define Led1 LATAbits.LATA0 //LED1
#define Led2 LATAbits.LATA1 //LED2
#define Led3 LATAbits.LATA2 //LED3
#define Led4 LATAbits.LATA3 //LED4
#define Led5 LATAbits.LATA5 //LED5
```

[14] Considerou-se que o aluno já possui conhecimento sobre o procedimento de criação de projetos no ambiente MPLAB®. Item abordado no capítulo 5.

```
#define Led6 LATAbits.LATA6 //LED6
#define Led7 LATAbits.LATA7 //LED7

main(){

    // Reset
    LATA = 0;
    LATB = 0;
    LATC = 0;
    LATD = 0;
    LATF = 0;
    LATG = 0;

    // Configuração da direção dos pinos de I/O's. (0-Output 1-Input)
    DDPCONbits.JTAGEN = 0;

    TRISA = 0xFF80; // 1111 1111 1000 0000 Leds: PORT A0-A6 (Output), Botao_3: PORT A7 (Input)
    TRISB = 0xFFFF;
    TRISC = 0xFFFF;
    TRISD = 0xEFCF; // 1110 1111 1100 1111 Botoes: PORT D6,D7,D13 (Input)
    TRISE = 0xFF00;
    TRISF = 0xFFFF;
    TRISG = 0xFEBF;


    while(1) {
        if(!Botao_1) Led1 = 1; //TESTE BOTÃO 1
        else Led1 = 0;
        if(!Botao_2) Led2 = 1; //TESTE BOTÃO 2
        else Led2 = 0;

        if(!Botao_3) Led3 = 1; //TESTE BOTÃO 3
        else Led3 = 0;

        if(!Botao_4) Led4 = 1; //TESTE BOTÃO 4
        else Led4 = 0;
    }
}
```

Compile o programa (**Ctrl+F10**).

Perceba que ao abrir o MPLAB® SIM apareceu um menu, denominado *debug*, com 8 botões a saber: *Run*, *Halt*, *Animate*, *Step Into*, *Step Over*, *Step Out*, *Reset* e *Breakpoints*. (Figura 80)

Figura 80 Menu debug para o MPLAB® SIM

Por instante vamos analisar o botão de *Breakpoint*.

Breakpoints são paradas intencionais inseridas ao longo da seqüência de execução do programa cuja finalidade é verificar se o programa esta passando por determinada linha de código ou simplesmente fazer uma pausa em algum ponto para verificação do estado de *flags*, variáveis, registradores, etc.

No MPLAB® existem 3 formas de se adicionar um *breakpoint*: Clicando duas vezes com o botão da esquerda sobre a linha na qual se queira inserir o *breakpoint*; Clicando com o botão direito sobre a linha e selecionando o item ***'Set Breakpoint'*** ou ainda colocando o cursor do *mouse* sobre a linha e clicando no ícone de *Breakpoint* no menu mostrado na Figura 80.

Para verificar seu funcionamento faça duas simulações: Na primeira simplesmente clique no botão *Run* apresentado na Figura 80 e perceba que o programa fica em execução constante (Figura 81) mas aparentemente nada acontece.

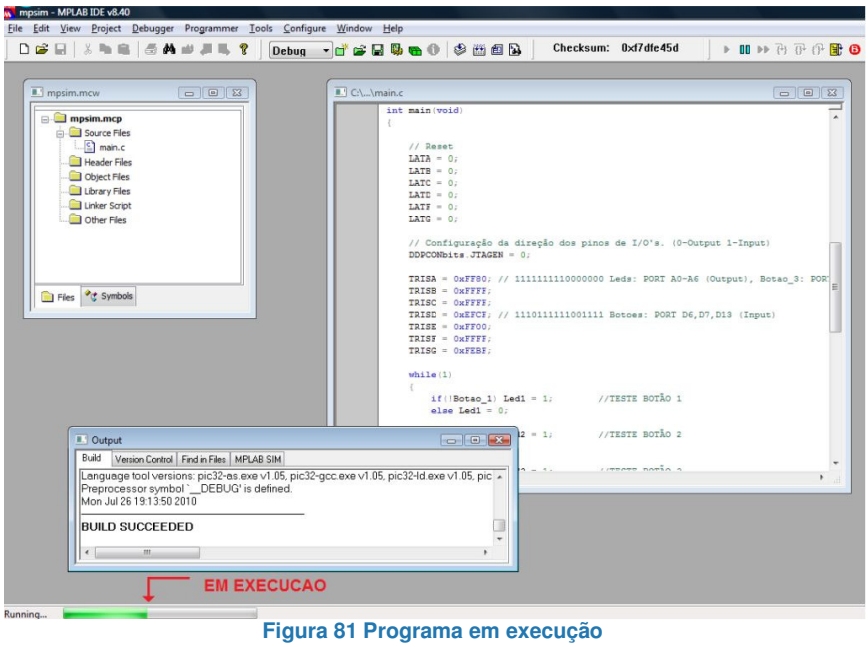

Figura 81 Programa em execução

Para parar a simulação pressione o botão *Halt*. Perceba que uma seta verde irá aparecer em alguma linha do código. Essa seta indica o ponto no qual o programa parou sua execução. (Figura 82)

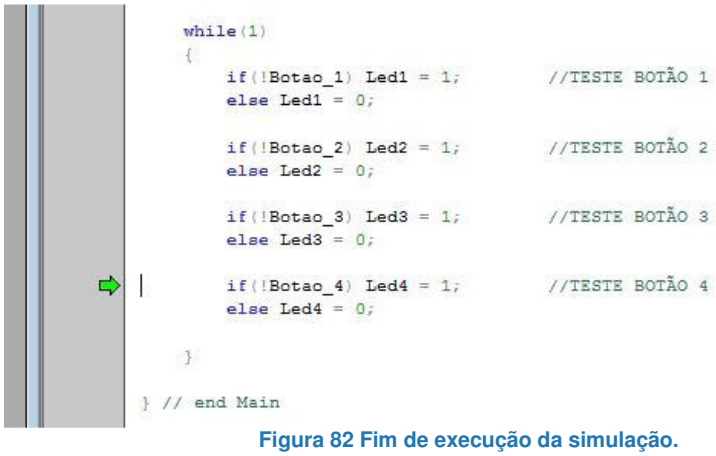

Figura 82 Fim de execução da simulação.

Na segunda simulação inclua quatro *breakpoints*, um para cada teste de acionamento dos botões. (Figura 83)

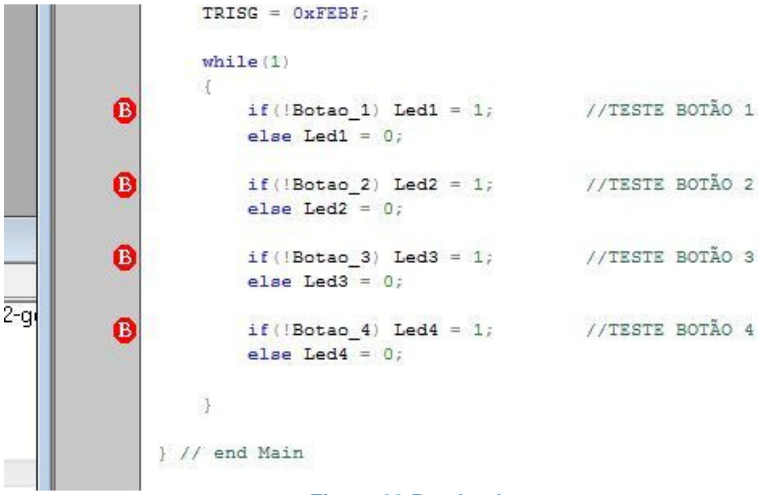

Figura 83 Breakpoints

Pressione o botão *Run* e, cada vez que o programa parar sua execução, pressione novamente o botão *Run*. Perceba que, sempre que o programa para a execução, uma seta verde aparece indicando o ponto a partir do qual o programa irá recomeçar.

Compreendido o funcionamento de um *breakpoint* passaremos à análise do botão *Step Into* com o qual é possível percorrer cada linha de código. Coloque um *breakpoint* na primeira linha do programa, clique no botão Run e assim que o programa parar sua execução (seta verde) pressione o botão *Step Into* continuamente. (Figura 84)

Perceba que depois de entrar no *loop while* o programa sempre entra na linha *if* mas nunca entra na linha *else*, a explicação desse fato será dada a seguir.

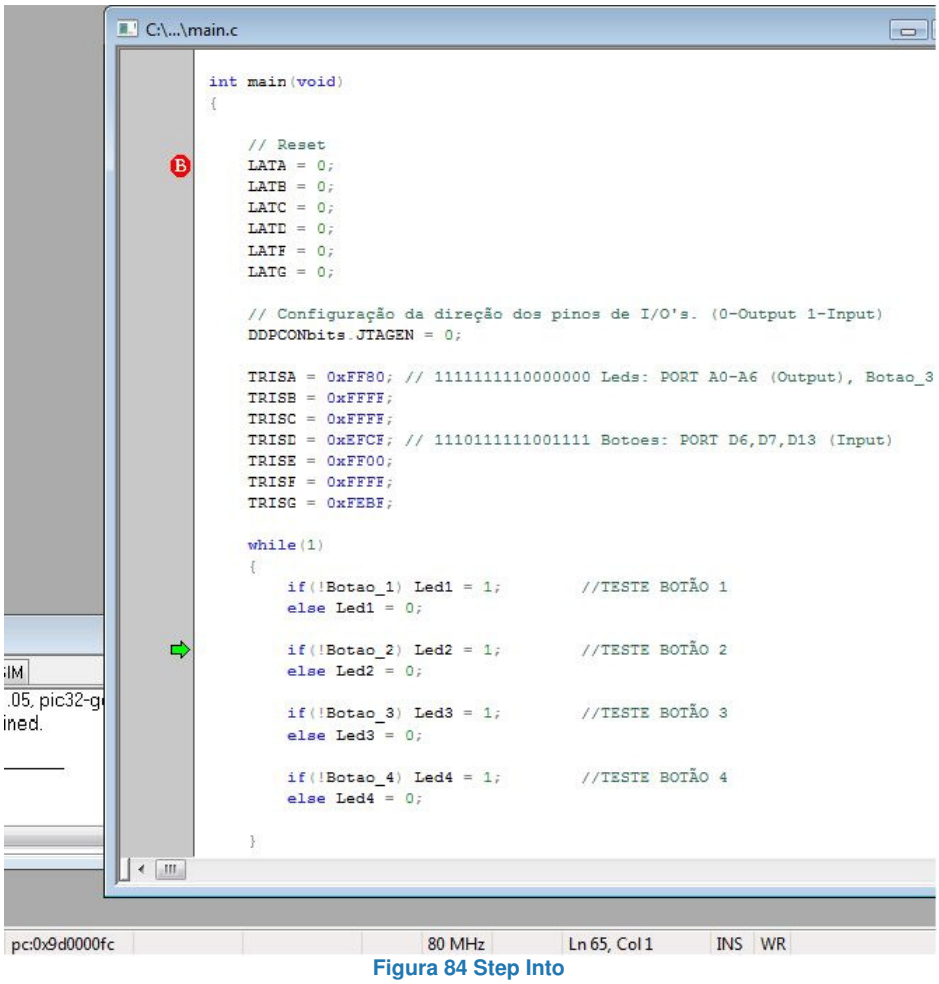

Figura 84 Step Into

Basicamente o programa não entra na linha *else* porque as variáveis Botao_1, Botao_2, Botao_3 e Botao_4 estão com o valor '0' e, ao se fazer a negação, viram '1' na expressão avaliada pelo *if* e por isso o programa nunca entra na linha *else*.

Desta forma podemos inferir que os PORTs RD6, RD7, RA7 e RD13, representados pelos botões, estão com nível lógico zero e os PORT RA0-RA3, representados pelos LEDs, estão em nível lógico '1' haja visto que o programa sempre entra na linha *if*.

Watchpoint

Para comprovar o que foi dito podemos utilizar um recurso oferecido pelo MPLAB® SIM no qual é possível verificar o estado de todos os registradores do PIC bem como as variáveis declaradas no programa. Este recurso é denominado *Watchpoint*.

Para abrir o *watch window* vá em: ***"View >> Watch"***.

Na janela que se abriu existem dois campos. Com o da direita (*Add SFR*) é possível adicionar os registradores que se queira observar. Com o campo da esquerda (*Add Symbol*) é possível adicionar as variáveis criadas no programa.

Por instante selecione PORTA e PORTD em Add SFR (*Add Special Function Registers*) para visualizar o estado dos bits correspondentes aos botões e LEDs. Clicando com o segundo botão do mouse é possível inserir a visualização em binário. (Figura 85)

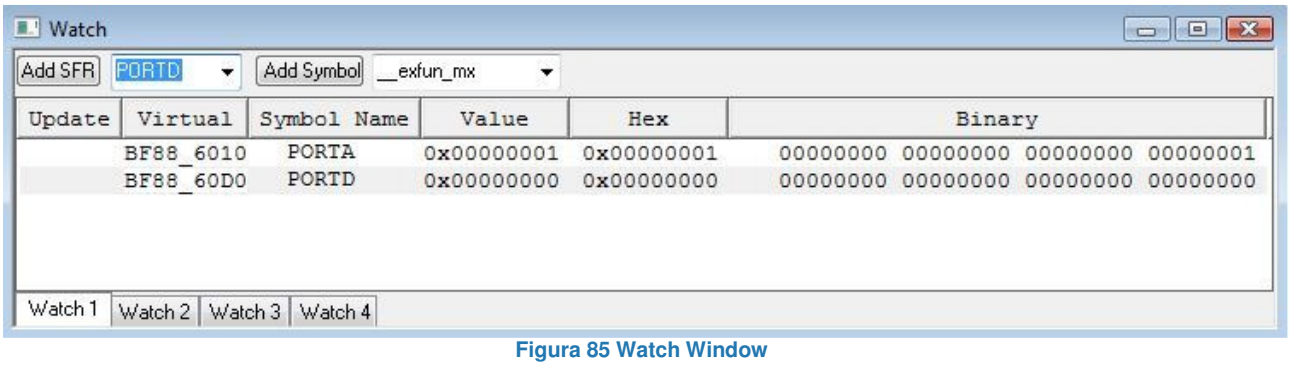

Figura 85 Watch Window

Agora pressione o botão *Reset* e em seguida *Run.* Seu programa deverá parar na primeira linha de código devido ao *breakpoint*.

Pressione o botão *Step Into* até entrar no *loop while*.

Conforme dito anteriormente os *bits* 6, 7 e 13 do PORTD e o *bit* 7 do PORTA estão em nível lógico zero. Isso faz com que, ao passar pelas lógicas dos *if* os *bits* 0, 1, 2 e 3 do PORTA fiquem em nível um. (Figura 86)

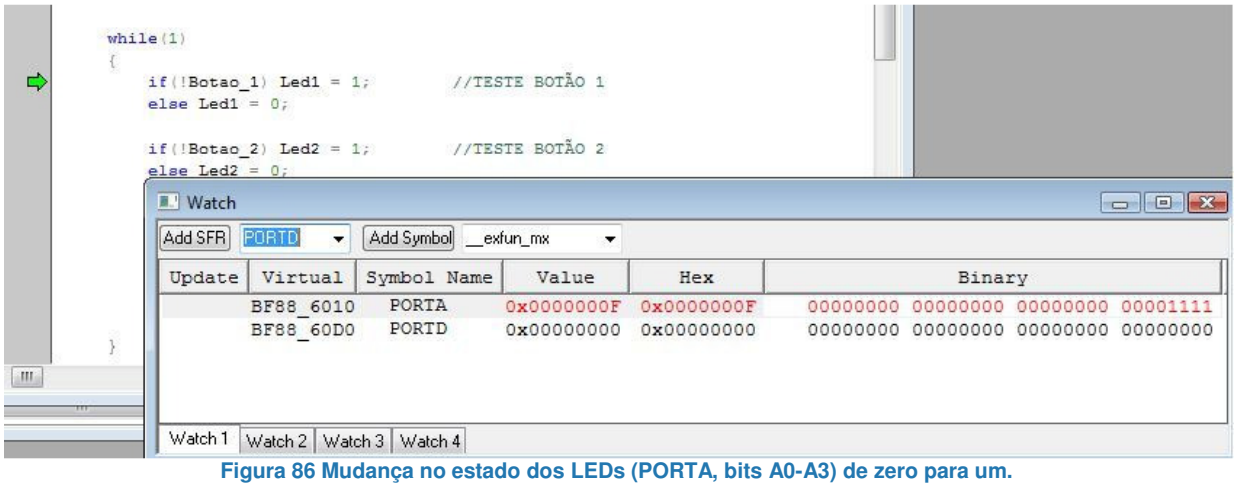

Figura 86 Mudança no estado dos LEDs (PORTA, bits A0-A3) de zero para um.

Para mudar o estado de um dos botões devemos gerar um estímulo e alterar seu estado. Por exemplo, para apagar o LED 1 devemos fazer com que o PORTD6, representado pelo Botao_1, assuma o valor um.

Estímulo

Existem dois tipos de estímulo, assíncrono ou síncrono. O estímulo assíncrono pode ser disparado pelo usuário a qualquer instante. O síncrono é disparado a partir de algum evento ocorrido ou em certos instantes de tempo pré-programados pelo usuário.

Para alterar o estado dos botões iremos utilizar o evento assíncrono. Para isso vá em ***"Debugger >> Stimulus >> New Workbook"***. Na janela que se abriu vá na aba ***"Asynch"*** e inclua os PORTs, referentes aos quatro botões, na coluna **Pin/SFR**. Na coluna *action* selecione *Toggle*, assim toda ver que for disparado um estimulo o valor da porta correspondente irá se alternar. (Figura 87)

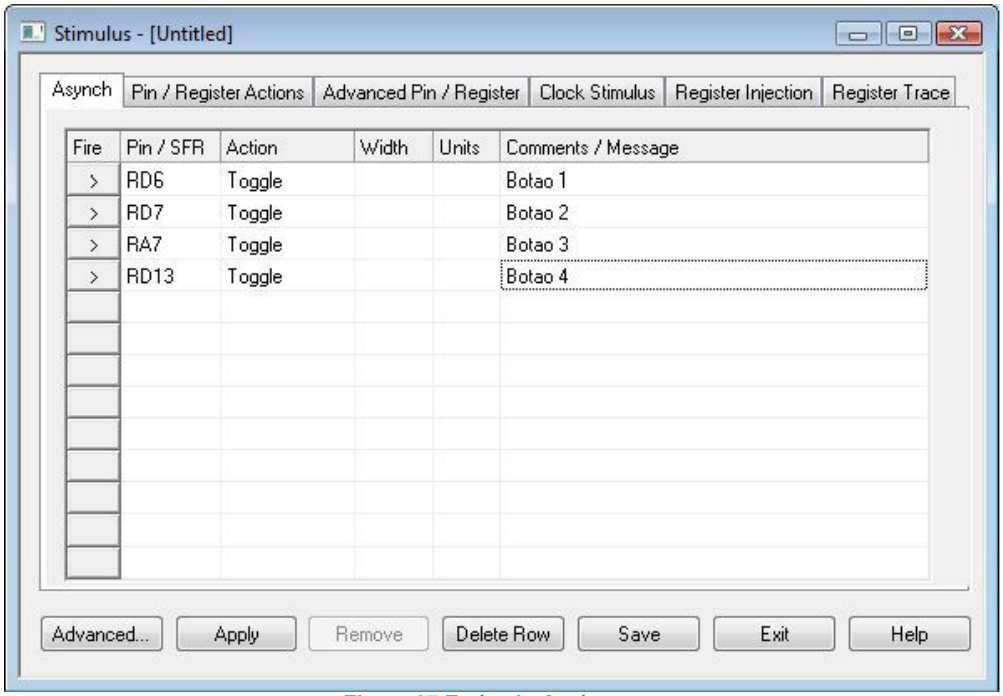

Figura 87 Estímulo Assíncrono

Agora clique em *Animate* no menu do *debug* do MPLAB SIM (Figura 80).

Perceba que a seta verde percorre os quatro testes *if* e o valor dos leds (PORT A0-A3) continuam em 1.

Caso você queira trocar o estado de qualquer um dos quatro PORTs que representam os botões basta clicar na seta logo a esquerda do PORT desejado mostrado na Figura 87.

Como exemplo clique na seta que representa o PORT RD7. Perceba que uma mensagem aparece na janela de *output* (Figura 88) indicando que o estímulo foi dado.

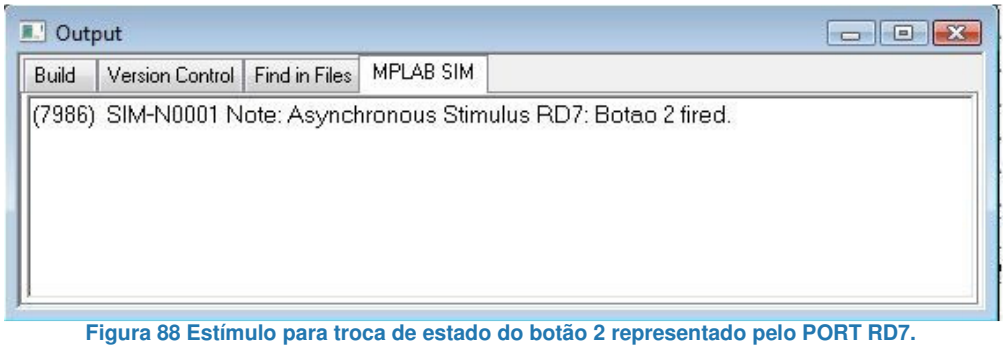

Figura 88 Estímulo para troca de estado do botão 2 representado pelo PORT RD7.

Além disso o estado do PORT RD7 (Botão 2) foi alterado de zero para um e o estado do PORT RA1 (LED 2) foi alterado de um para zero. (Figura 89)

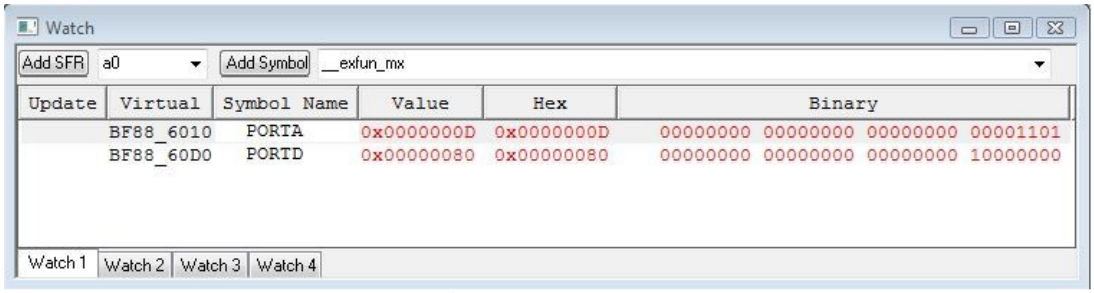

Figura 89 Alteração do estado dos PORTs RD7 e RA1.

Agora lance vários estímulos utilizando os PORTS RD6, RD7, RA7 e RD13 e verifique o valor dos bits nos PORTs 'D' e 'A'. (Figura 90)

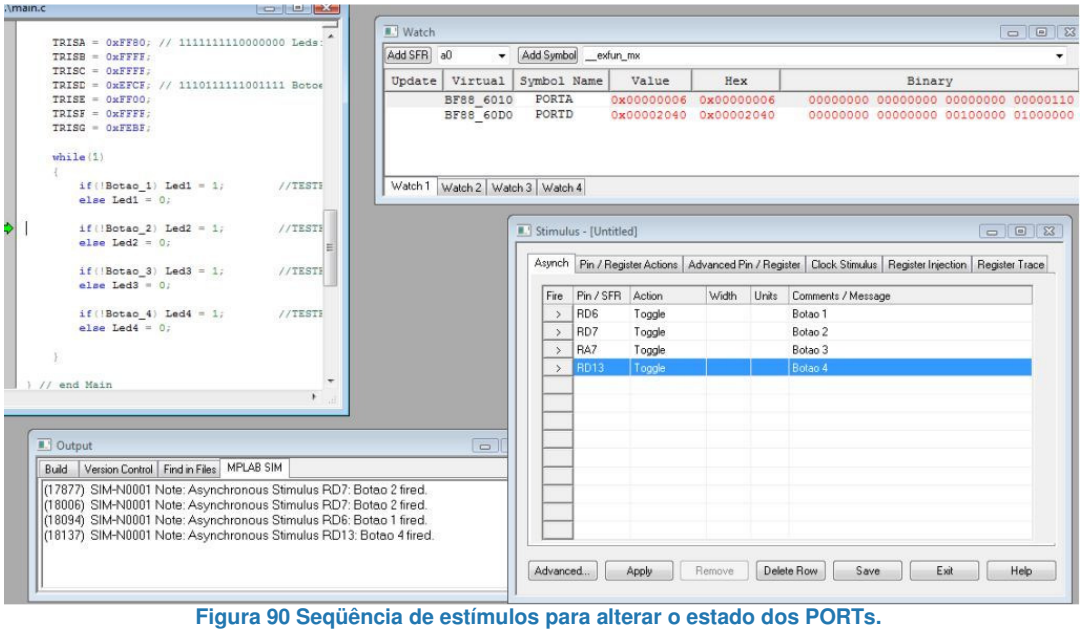

Figura 90 Seqüência de estímulos para alterar o estado dos PORTs.

Caso queira você poderá salvar os estímulos pré-definidos (Figura 87) e utilizá-los em aplicações futuras.

Como dito anteriormente com o *Watch Window* (Figura 85) é possível visualizar tanto os registradores do PIC quanto o valor das variáveis declaradas no programa. Para compreender como isso funciona altere o código da função ***main*** anterior por:

```
int main(void) {

    int i=0;
    int j=10000;

    // Reset
    LATA = 0;
    LATB = 0;
    LATC = 0;
    LATD = 0;
    LATF = 0;
    LATG = 0;
```

```
    // Configuração da direção dos pinos de I/O's. (0-Output 1-Input)
    DDPCONbits.JTAGEN = 0;

    TRISA = 0xFF80; // 1111111110000000 Leds: PORT A0-A6 (Output), Botao_3: PORT A7 (Input)
    TRISB = 0xFFFF;
    TRISC = 0xFFFF;
    TRISD = 0xEFCF; // 1110111111001111 Botoes: PORT D6,D7,D13 (Input)
    TRISE = 0xFF00;
    TRISF = 0xFFFF;
    TRISG = 0xFEBF;

    while(1)
    {
        i++;
        j--;
    }
} // end Main
```

Perceba que o código inicializa a variável 'i' com o valor '0' e a variável 'j' com o valor 10000. A cada ciclo do *loop while* o valor da variável 'i' é incrementado e o valor da variável 'j' é decrementado.

Para visualizar o programa compile-o (*Build All*), abra o *watch window* e acrescente as variáveis 'i' e 'j' através do campo *Add Symbol*. (Figura 91)

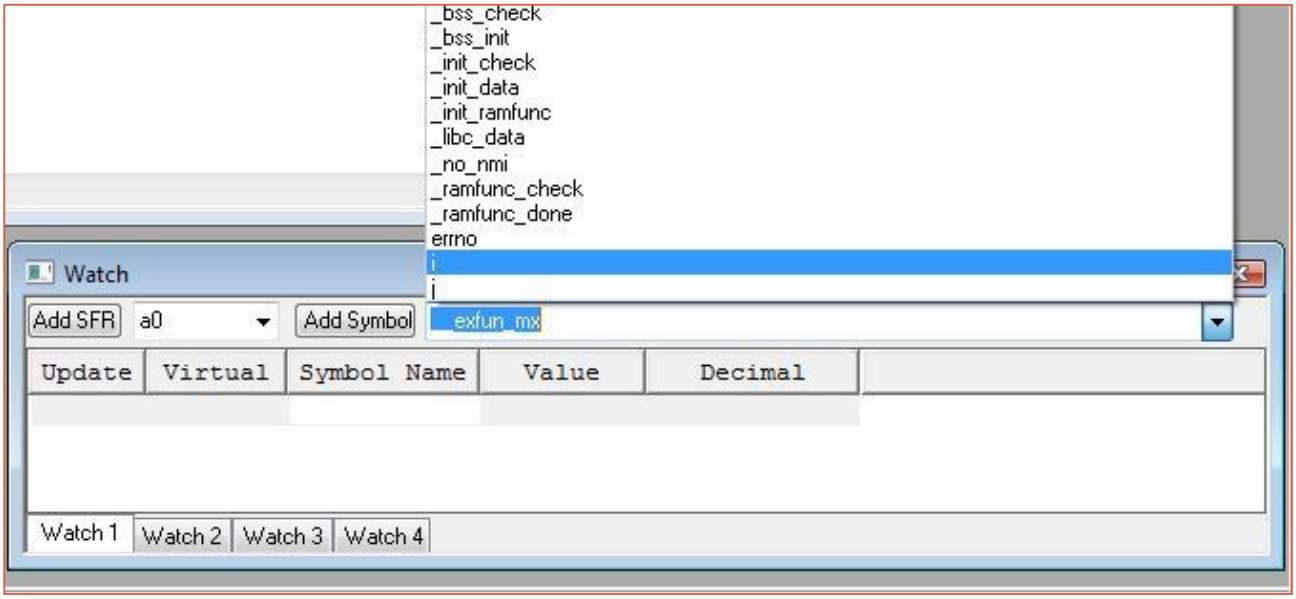

Figura 91 Visualizando variáveis através do Watch Window

A mensagem *'Out of scope'* deverá aparecer. (Figura 92) Isso é normal haja visto que não rodamos o programa ainda e dessa forma as variáveis 'i' e 'j' ainda não foram declaradas.

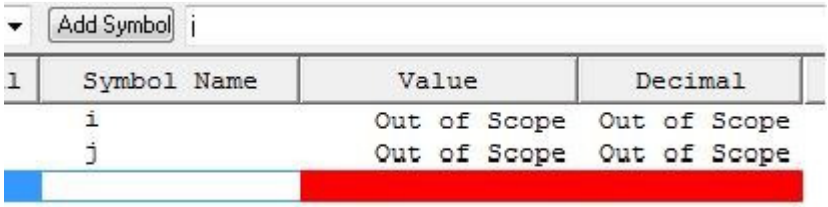

Figura 92 Mensagem devido a não inicialização das variáveis

Insira um *breakpoint* acima da linha do *loop while* e rode o programa (*Run*). Assim que o programa parar no *breakpoint* as variáveis já terão sido declaradas e dessa forma seu valor de inicialização poderá ser visto no *watch window*. (Figura 93)

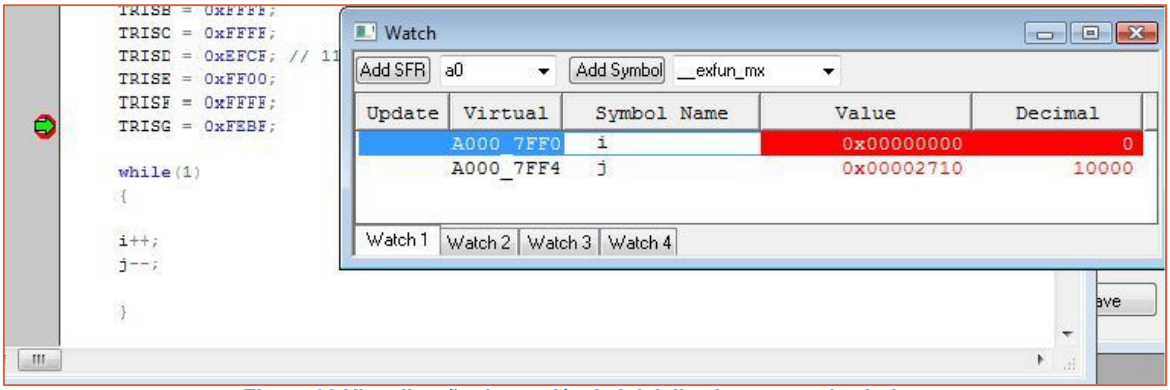

Figura 93 Visualização das variáveis inicializadas no watch window

Agora rode o programa passo a passo (*Step Into*) e veja o incremento da variável 'i' e decremento da variável 'j'. (Figura 94)

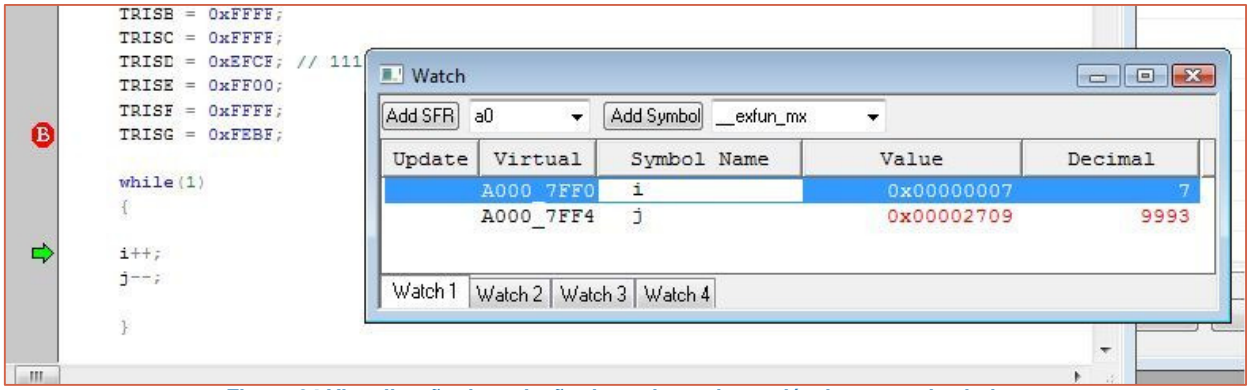

Figura 94 Visualização da variação dos valores das variáveis no watch window

Experimente alterar o valor das variáveis através do teclado. Clique duas vezes sobre o valor decimal da variável desejada e coloque algum número. Por exemplo vamos colocar o valor 2000 na variável 'i'. (Figura 95)

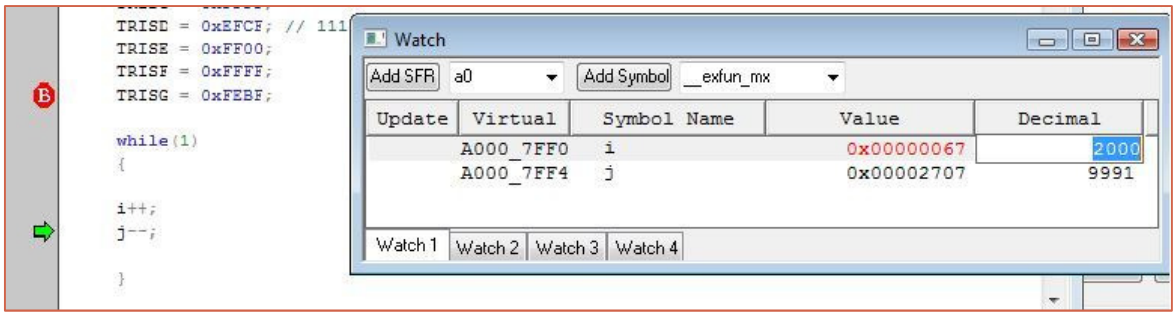

Figura 95 Alteração manual do valor das variáveis

Logic Analyzer

Além do *Watch Window* podemos visualizar os registradores do PIC utilizando uma ferramenta denominada *Logic Analyzer*. Diferentemente do *Watch Window* que apresenta somente o valor das variáveis o *Logic Analyzer* mostra a respectiva forma de onda.

Antes de abrir o *Logic Analyzer* vá em: *"Debugger >> Settings..."* e na aba *"Osc/Trace"* selecione a opção *"Trace All"*. (Figura 96) Esta opção permite termos acesso a lista de pinos do PIC.

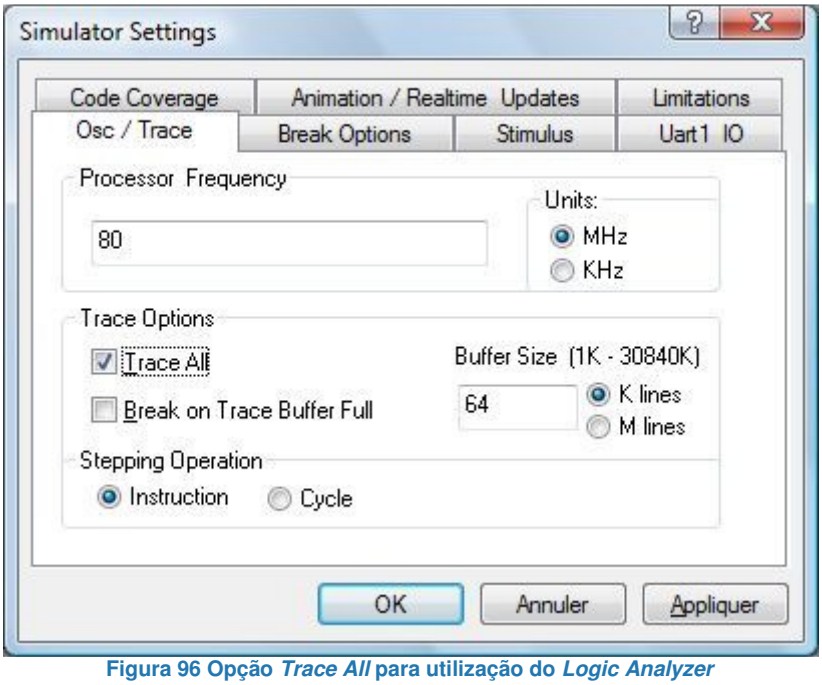

Figura 96 Opção *Trace All* para utilização do *Logic Analyzer*

Para abrir o *Logic Analyzer* vá em: ***"View >> Simulator Logic Analyzer"***. (Figura 97)

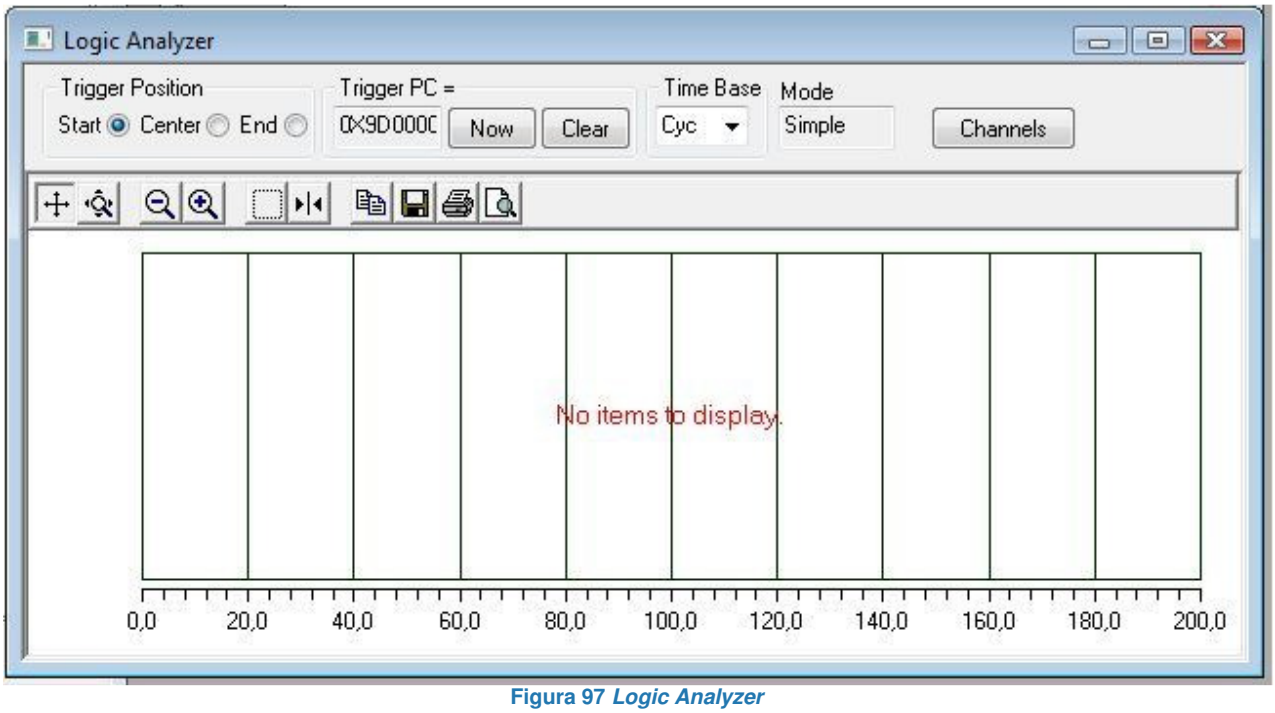

Figura 97 *Logic Analyzer*

No botão *channels* insira os canais RD6, RD7, RA7 e RD13. (Figura 98) Perceba que estes PORTS foram os mesmos utilizados para verificar o funcionamento da ferramenta *Watch Window* e gerar os estímulos.

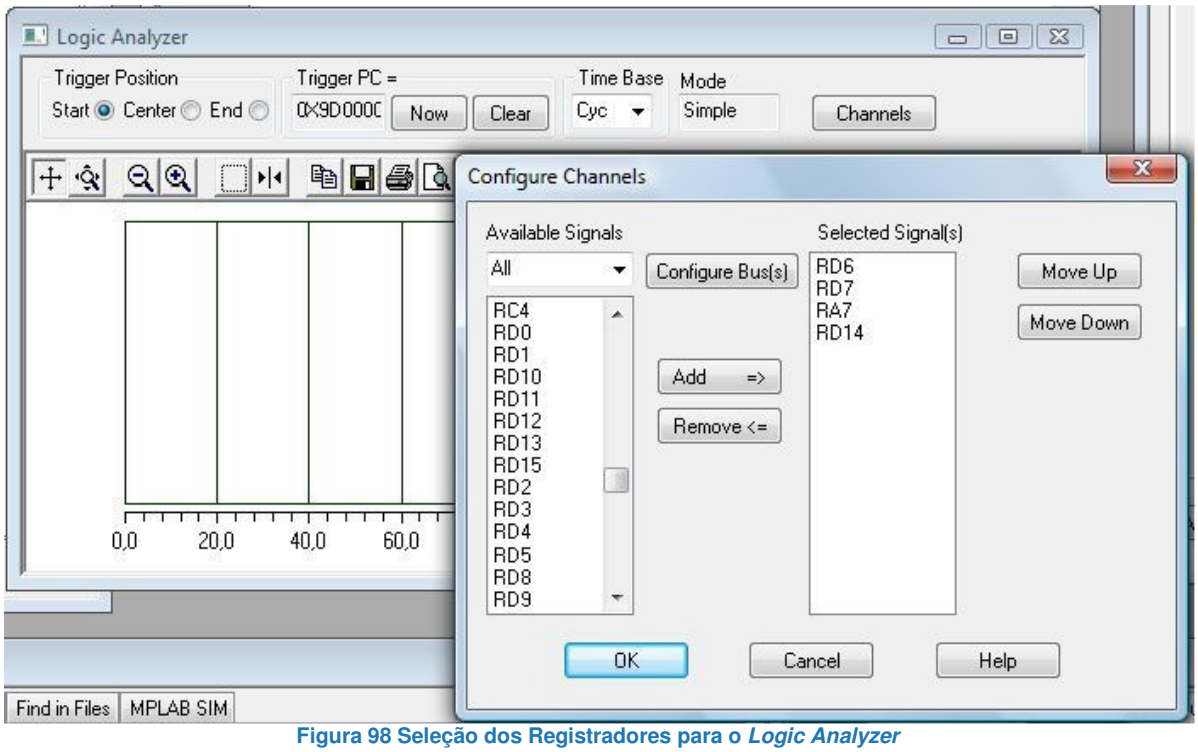

Figura 98 Seleção dos Registradores para o *Logic Analyzer*

Clique em *Animate* no menu do *debug* do MPLAB SIM (Figura 80).

Lance vários estímulos assíncronos aleatórios da mesma forma quando o *Watch Window* foi apresentado. Perceba, na janela do *Logic Analyzer*, que o valor dos PORTs RD6, RD7, RA7 e RD13 irá se alterar entre os níveis lógicos alto e baixo. (Figura 99)

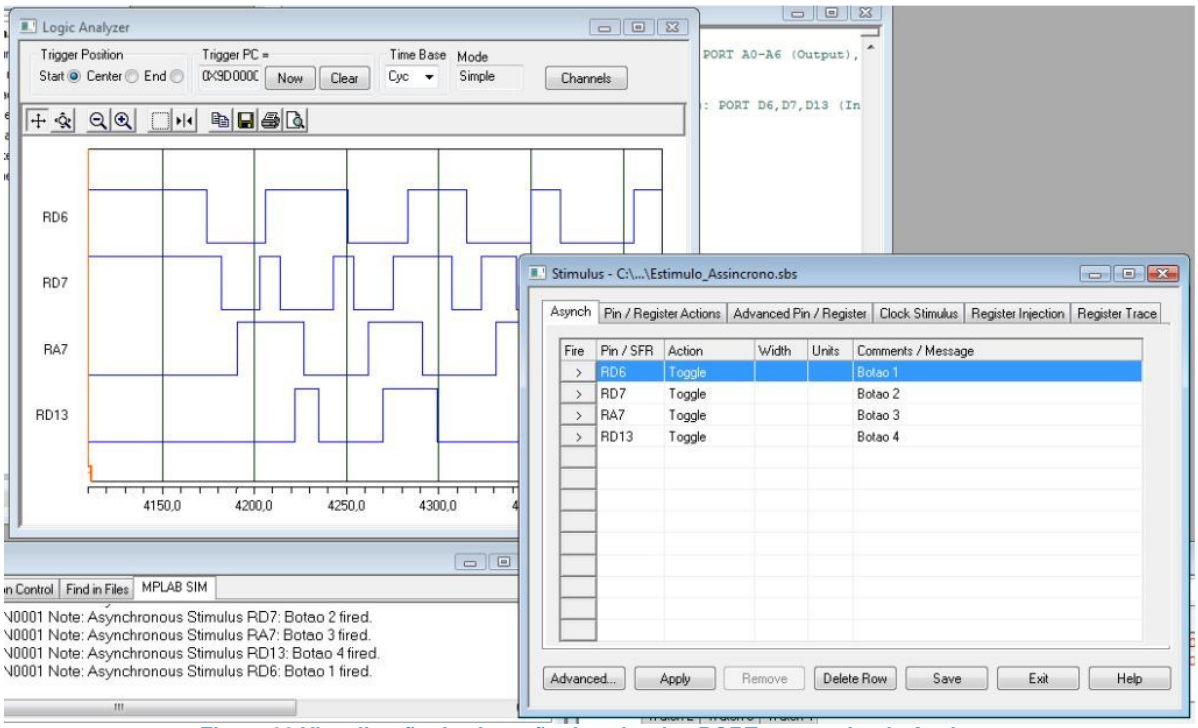

Figura 99 Visualização da alteração do valor dos PORTs com o *Logic Analyzer*

Para testar a UART insira o seguinte programa:

```
// INCLUDES
#include <p32xxxx.h> //include para o PIC32MX360F512


// CONFIGURACAO PARA GRAVACAO
#pragma config FPLLMUL = MUL_16, FPLLIDIV = DIV_2, FPLLODIV = DIV_8, FWDTEN = OFF
#pragma config POSCMOD = HS, FNOSC = PRI, FPBDIV = DIV_2


// INICIA SISTEMA
void init_sys(){
    // Reset
    LATA = 0;
    LATB = 0;
    LATC = 0;
    LATD = 0;
    LATF = 0;
    LATG = 0;

    // Configuração da direção dos pinos de I/O's. (0-Output 1-Input)
    DDPCONbits.JTAGEN = 0;

    TRISA = 0xFF80; // 1111111110000000 Leds: PORT A0-A6 (Output), Botao_3: PORT A7 (Input)
    TRISB = 0xFFFF;
    TRISC = 0xFFFF;
    TRISD = 0xEFCF; // 1110111111001111 Botoes: PORT D6,D7,D13 (Input)
    TRISE = 0xFF00;
    TRISF = 0xFFFF;
    TRISG = 0xFEBF;
}

// FUNCOES UART

void inicializa_UART1(void)
{
    // Se BRGH=0 -> U2BRG = (Fosc/FPBDIV)/(16*BaudRate) - 1
    // Se BRGH=1 -> U2BRG = (Fosc/FPBDIV)/(4*BaudRate) - 1
    // BaudRate = 19200bps
    U2BRG = 51; // BRGH=1 -> U2BRG = (8000000/2)/(4*19200) - 1

    U1MODEbits.UARTEN = 1; // Habilita UART2
    U1MODEbits.UEN = 0; // TX e RX habilitados, CTS e RTS controlados via hardware (MAX232)
    U1MODEbits.ABAUD = 0; // Desabilita o autobaudrate
    U1MODEbits.BRGH = 1; // Configuração do BRGH
    U1MODEbits.PDSEL = 0; // 8 bits de dados, sem paridade

    U1MODEbits.STSEL = 0; // 1 stop bit

    U1STAbits.UTXISEL0 = 0; // gera interrupcao a cada Tx
    U1STAbits.UTXISEL1 = 0;

    U1STAbits.URXISEL = 1; // gera interrupcao a cada Rx

    U1STAbits.UTXEN = 1; // Habilita pino TX
    U1STAbits.URXEN = 1; // Habilita pino RX
}

unsigned int RxUART1(void) // Lê UART
{
    if(U1MODEbits.PDSEL == 3) return (U1RXREG);
    else return (U1RXREG & 0xFF);
}

void TxUART1(unsigned int data) // Escreve UART
{
    if(U1MODEbits.PDSEL == 3) U1TXREG = data;
    else U1TXREG = data & 0xFF;
}


int main(void)
{
    char caractere;
```

```
    init_sys(); // Inicializa o sistema

    inicializa_UART1();

    putsUART1("UFMG\r\n");
    putsUART1("COMUNICACAO RS232\r\n");

    while(1);

}
```

A UART foi configurada com uma taxa de 19200 bps para aumentar a velocidade de simulação do código. Caso tivéssemos usado uma taxa de 9200 bps a simulação iria se tornar muito lenta.

Para visualizar a saída do programa vá em ***"Debugger >> Settings..."*** e na aba ***"Uart1 IO"*** selecione a opção ***"Enable Uart1 IO"*** e no campo ***"Output"*** selecione ***"Window"***. Assim os dados enviados para a UART pelo seu programa irão aparecer na janela ***"Output"***. (Figura 100)

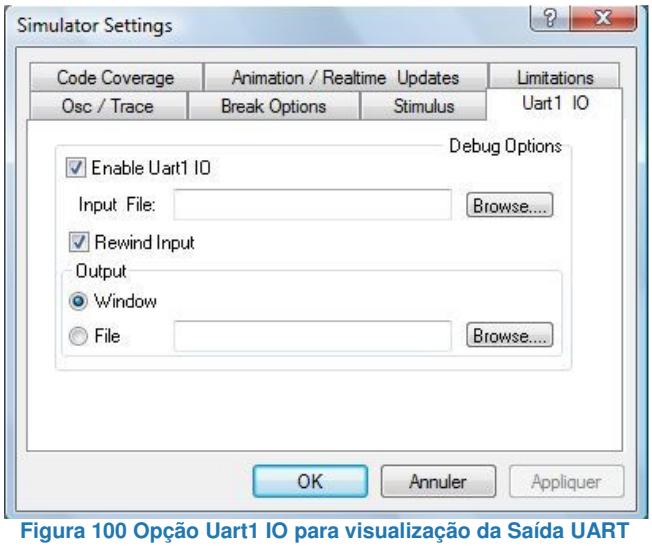

Figura 100 Opção Uart1 IO para visualização da Saída UART

A saída do programa é mostrada na Figura 101.

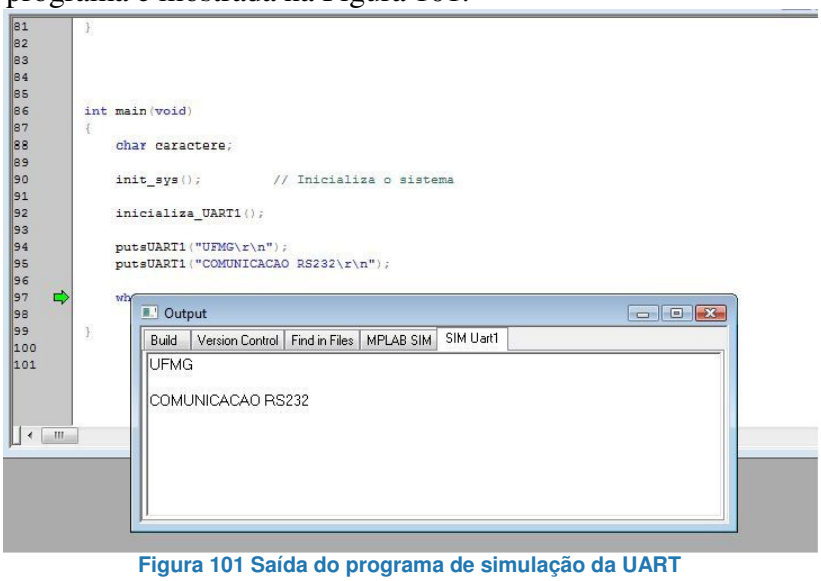

Figura 101 Saída do programa de simulação da UART

Stopwatch

Para finalizar a apresentação dos recursos oferecidos pelo MPLAB® SIM vamos analisar o **Stopwatch**. Com esse recurso é possível mensurar o tempo real gasto pelo PIC para executar algum comando.

Primeiramente é preciso configurar o clock do periférico que será utilizado. Para isso vá em ***"Debugger >> Settings..."***, aba ***"Osc/ Trace"***, campo ***"Processor Frequency"*** e insira o clock de operação dos periféricos do PIC. (Figura 102)

ATENÇÃO: O clock de operação dos periféricos é diferente do clock real do PIC pois, devido às configurações do projeto, o clock do cristal pode ser dividivo por: 8, 4, 2 ou 1 vezes.

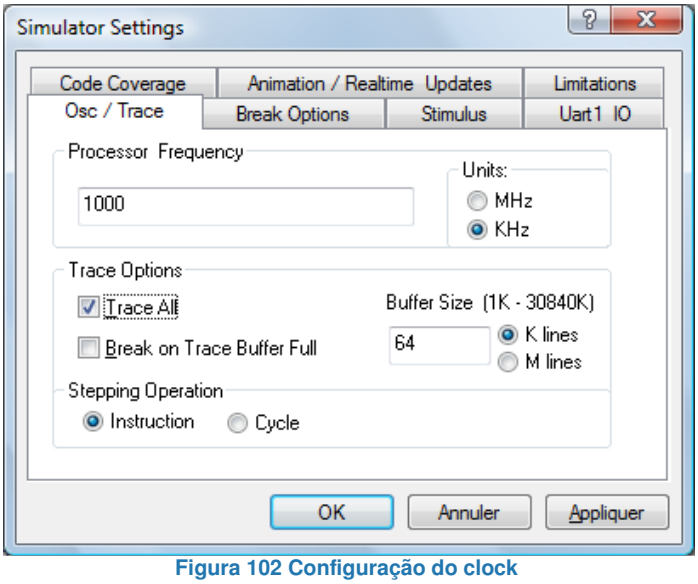

Figura 102 Configuração do clock

Ainda na aba **"Osc/Trace"** selecione a opção ***"Trace All"*** no campo ***"Trace Options"*** para monitorar todos os registradores do PIC. (Figura 102)

Para compreender melhor o funcionamento do recurso Stopwatch crie um novo projeto e insira o seguinte código na funcção main:

```
1.  // INCLUDES
2.  #include <p32xxxx.h>                    //include para o PIC32MX360F512
3.
4.  // CONFIGURACAO PARA GRAVACAO
5.  #pragma config FPLLMUL = MUL_16, FPLLIDIV = DIV_2, FPLLODIV = DIV_8, FWDTEN = OFF
6.  #pragma config POSCMOD = HS, FNOSC = PRI, FPBDIV = DIV_8 //(PBCLK is SYSCLK divided by 8,4,2,1)
7.
8.  void delay_1ms_x(unsigned int x)
9.  {
10.         T5CONbits.TON = 0;          // Timer5 desligado
11.         TMR5 = 0;                   // Zera o timer para início de contagem
12.
13.         // Timer5 Prescaler
14.         // TCKPS   ->   Prescaler
15.         //  0    ->   1:1
16.         //  1    ->   1:2
17.         //  2    ->   1:4
18.         //  3    ->   1:8
19.         //  4    ->   1:16
20.         //  5    ->   1:32
21.         //  6    ->   1:64
22.         //  7    ->   1:256
23.         T5CONbits.TCKPS = 3;        // Prescaler 1:8
24.
25.         // Configura o registrador de período
26.         // PR5 = (Fosc * Tempo)/(FPBDIV * PS)
27.         // PR5 = (8000000 * 0,001)/(8 * 8) = 125
```

```
28.         PR5 = 125;
29.
30.         T5CONbits.TCS = 0;              // Modo timer (clock interno)
31.         T5CONbits.TGATE = 0;
32.
33.           IFS0bits.T5IF = 0;            // Limpa o flag
34.
35.         T5CONbits.TON = 1;              // Timer5 ligado
36.
37.         while(x != 0)
38.         {
39.                     while(IFS0bits.T5IF == 0);      // Aguarda o tempo de 1 ms
40.                     IFS0bits.T5IF = 0;              // Limpa o flag
41.                     x--;                            // Decrementa x
42.         }
43.
44.         T5CONbits.TON = 0;                          // Desliga o Timer5
45.   }
46.
47.   int main(){
48.
49.         // Reset
50.         LATA = 0;
51.         LATB = 0;
52.         LATC = 0;
53.         LATD = 0;
54.         LATF = 0;
55.         LATG = 0;
56.
57.         // Configuração da direção dos pinos de I/O's. (0-Output 1-Input)
58.               DDPCONbits.JTAGEN = 0;
59.
60.         TRISA = 0xFF80; // 1111111110000000 Leds: PORT A0-A6 (Output), Botao_3: PORT A7 (Input)
61.         TRISB = 0xFFFF;
62.         TRISC = 0xFFFF;
63.         TRISD = 0xEFCF; // 1110111111001111 Botoes: PORT D6,D7,D13 (Input)
64.         TRISE = 0xFF00;
65.         TRISF = 0xFFFF;
66.         TRISG = 0xFEBF;
67.
68.
69.         while(1)
70.         {
71.                                 delay_1ms_x(2000);
72.         }
73.
74.   }
```

Perceba, na linha 6 do código acima, que o clock que alimenta os periféricos é dado pelo clock do PIC dividido por 8 (FPBDIV = DIV_8).Como o PIC em questão possui um cristal de 8MHz o clock do periférico será 8MHz/8=1MHz, dessa forma o valor passado para a configuração do clock é 1MHz (100Khz), observe a Figura 102.

Insira um Breakpoint na linha 71. (Figura 103)

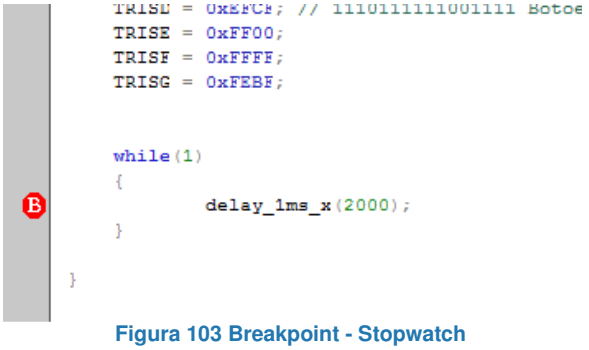

Figura 103 Breakpoint - Stopwatch

Agora vá em “Debugger >> Stopwatch” para abrir o recurso Stopwatch. Observe no parâmetro **“Processor Frequency”** que a freqüência de 1MHz setada anteriormente está selecionada. (Figura 104)

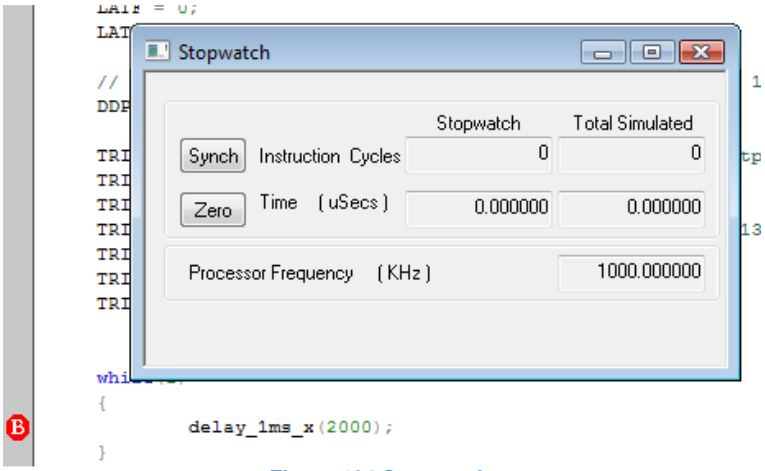

Figura 104 Stopwatch

Quando você iniciar a simulação (Figura 80) o cronômetro irá começar a contagem até parar no Breakpoint.

Os valores da janela do Stopwatch serão modificados conforme mostra a Figura 105. Observe queforam executadas 705 instruções até o breakpoint durante um intervalo de tempo de 705µS. O que resulta em uma instrução por micro-segundo, ou ainda, uma freqüência de $\frac{1}{10^{-6}} = 1MHz$, que é exatamente a freqüência pré-ajustada.

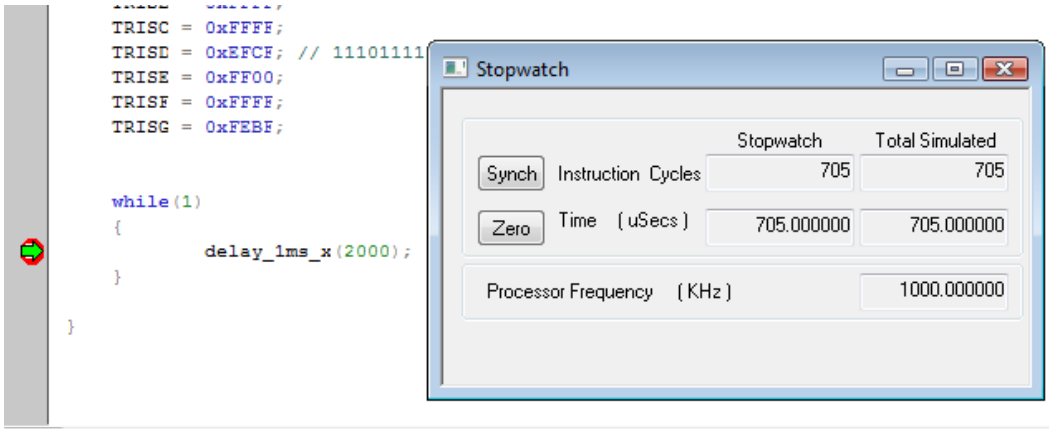

Figura 105 Stopwatch 2

Antes de prosseguir com a execução do programa limpe o contador selecionando a opção **“Zero”**. Perceba que ele irá zerar somente a coluna correspondente ao Stopwatch, mantendo o tempo total simulado (Total Simulated). (Figura 106)

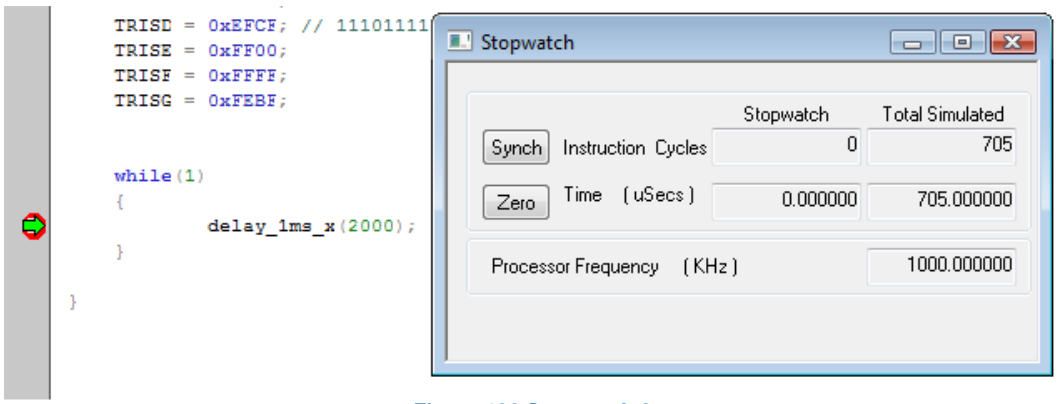

Figura 106 Stopwatch 3

Agora prossiga com a execução do programa. Observe no código que o programa irá gerar um *delay* de 2segundos. Após o fim dos 2 segundos o programa volta ao mesmo ponto e para novamente no breakpoint. Neste instante a janela do Stopwatch apresenta os valores mostrados na Figura 107.

Observe que o tempo decorrido para executar a função delay foi de 2.014065segundos. O valor não é exatamente 2 segundos porque, da forma na qual o código foi implementado, a dunção delay configura o timer para depois fazer a contagem, e essa configuração leva alguns ciclos de clock para ser realizada.

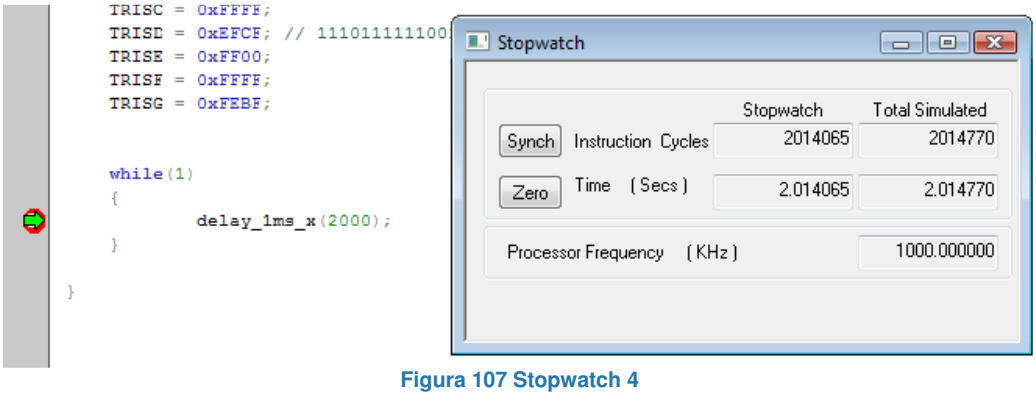

Figura 107 Stopwatch 4

Compreendido o funcionamento das ferramentas do MPLAB SIM e sua utilidade o programador poderá testar os programas antes de gravá-los no PIC.

ERRATAS

Errata I : Esquemático Leds e Botoes

O diagrama elétrico da ligação dos leds e botões pode ser obtido no arquivo ***"Esquemas Elétricos >> mainBoard >> P-CAD EDA – [Sheet1]"*** disponível com o CD fornecido com o Kit. Para facilitar a compreensão a ligação dos mesmos foi representada a seguir. (Figura 108 e Figura 109)

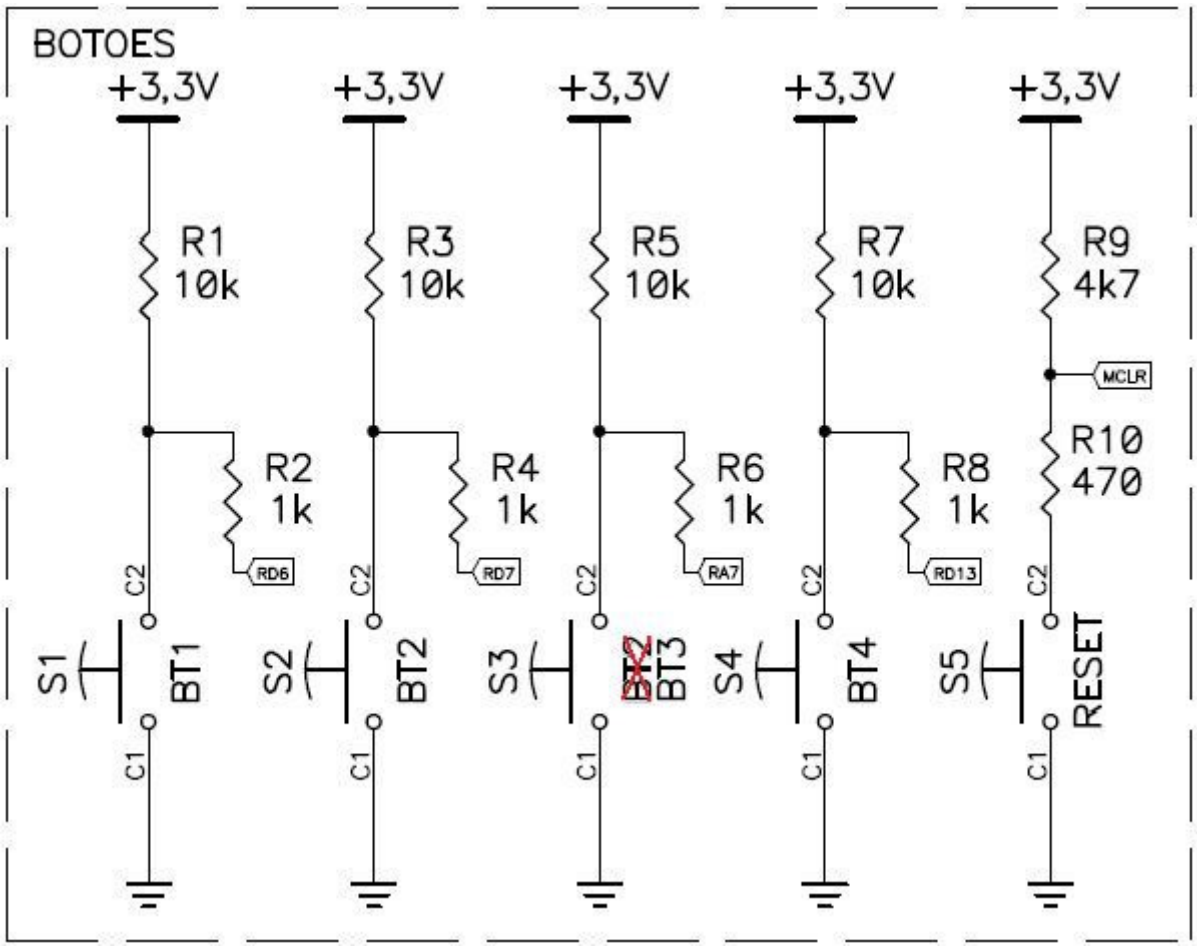

Figura 108 Esquema Elétrico dos Botes do kit EXPLORER16 BR

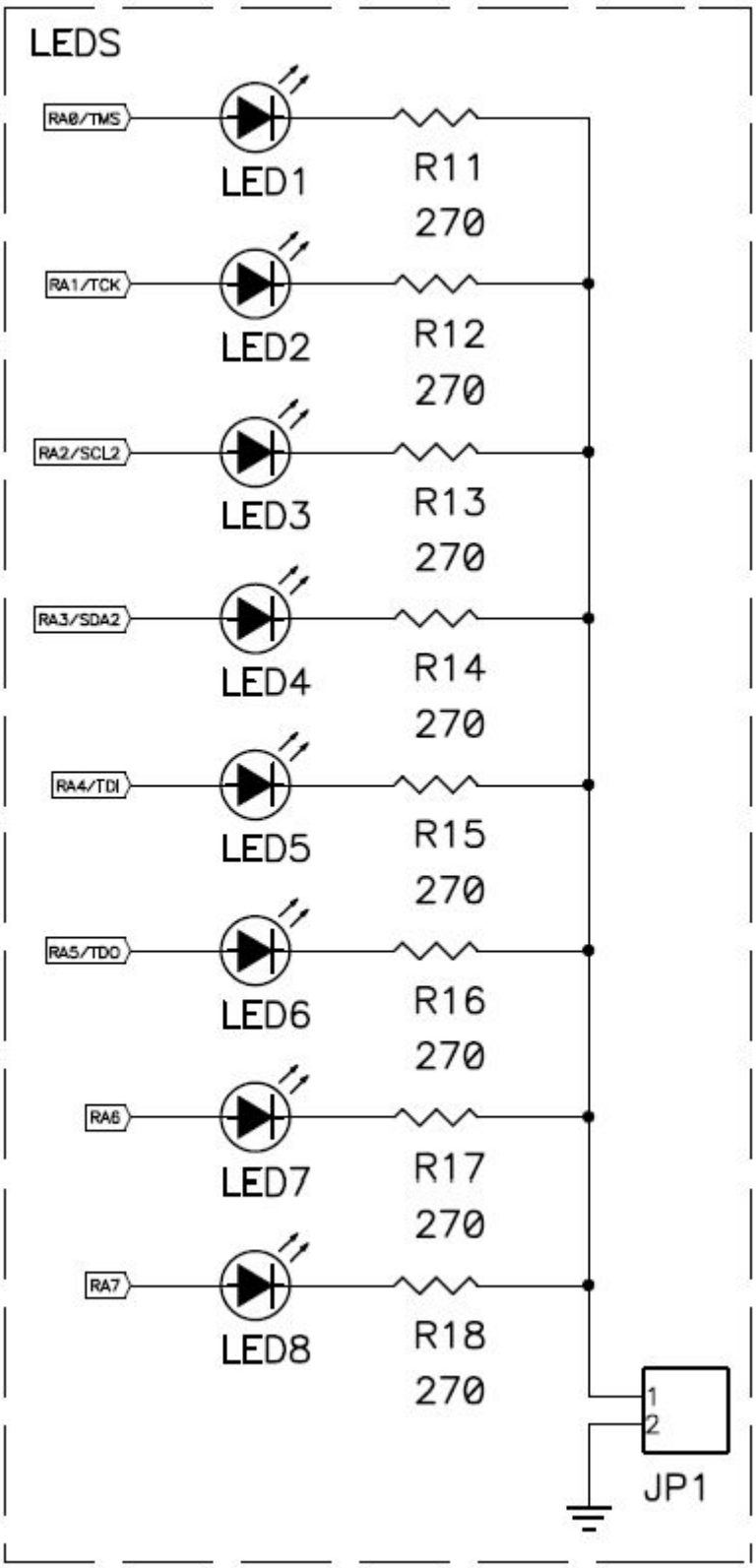

Figura 109 Esquema Elétrico dos Leds do kit EXPLORER16BR

Perceba que tanto o LED8 quanto a chave S3 estão ligados ao PORT RA7 e, para que os leds possam ser usados, o *jumper* 1 deve ser utilizado. O que resulta em um circuito equivalente segundo a Figura 110 a seguir.

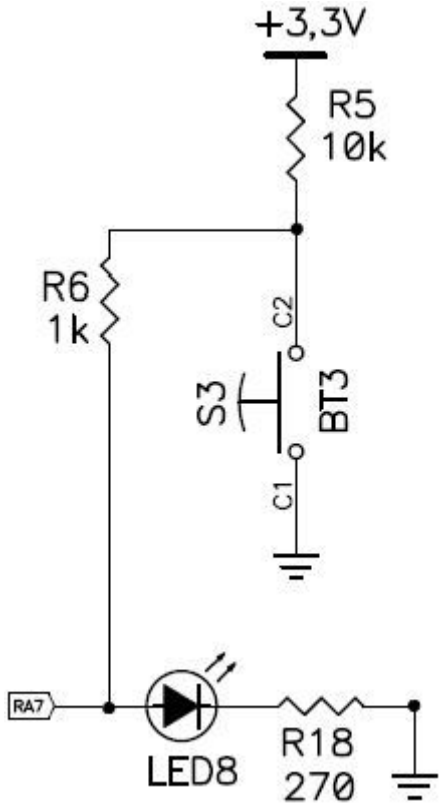

Figura 110 Circuito resultante para o PORT RA7

Perceba que sempre haverá um circuito entre a alimentação (3.3 V) e o circuito do LED8. Assim, sempre haverá uma pequena corrente circulando pelo LED8 e este NUNCA ESTARÁ TOTALMENTE APAGADO. Isso pode ser confirmado zerando-se todas as saídas para os leds e verificando que somente o LED8 apresenta um pequeno brilho, devido a esta corrente no circuito equivalente.

Como solução iremos manter o *jumper* para podermos acionar os leds de 1 a 7 e utilizaremos o PORT RA7 como entrada, ou seja, como chave S3, desconsiderando-se assim o uso do LED8.

Errata II : Efeito de carga do ICD2BR

Ao compilar o programa para acionamento dos leds percebeu-se que quando a placa ICD2 BR estava conectada ao kit EXPLORER16 BR os leds não acendiam devido a um efeito de carga. Para solucionar este problema deve-se desconectar co cabo RJ12 que liga a placa ICD2 BR ao kit EXPLORER16 BR e testar o programa.

Referências Bibliográficas

Atmel. "Atmel Corporation - Industry Leader in the Design and Manufacture of Advanced Semiconductors." Março 2010. http://www.atmel.com.

CCS. "CCS, Inc. - CCS C Compilers." Março 2010. http://www.ccsinfo.com/content.php?page=compilers.

Hantronix, Inc. ":: Welcome to HANTRONIX, Inc. ::." Maio 2010. http://www.hantronix.com/down/char-comm.pdf.

Hilgrave. *Hilgrave - Hyper Terminal HyperACCESS and DropChute communication software - Hilgrave.* Julho 2010.

HI-TECH. "Embedded C Compilers and Tools for Software Development: HI-TECH Software." Março 2010. http://www.htsoft.com/products/.

Labtools. "LabTools - PICs, Kits Didaticos, Treinamentos, Componentes Eletronicos." Março 2010. http://www.labtools.com.br.

MicroC. "mikroC - Advanced C compiler for PIC microcontrollers." Março 2010. http://www.mikroe.com/en/compilers/mikroc/pic/.

Microchip. "Microchip Technology Inc." Março 2010. http://www.microchip.com.

Mplab C18. "MPLAB C Compiler for PIC18 MCUs." Março 2010. http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en010014.

Mplab C30. "MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs." Março 2010. http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en010065.

Mplab C32. "MPLAB C Compiler for PIC32 MCUs." Março 2010. http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2615&dDocName=en532454.

Mplab IDE. "MPLAB Integrated Development Environment." Março 2010. http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002.

National Instruments. "National Instruments - Test and Measurement." Março 2010. http://www.nationalinstruments.com.

Texas Instruments. "Analog, Embedded Processing, Semiconductor Company, Texas Instruments." Março 2010. http://www.ti.com.